

การเขียนโปรแกรมภาษาซี

โดย

วิจักขณ์ ศรีสังจะเลิศวาจา
ดุษฎี ประเสริฐธิติพงษ์

ภาควิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่

พฤษภาคม 2545

สารบัญ

| | |
|-----------------------------------|----|
| บทที่ 1. แนะนำภาษาซี | 1 |
| 1.1 แนะนำความเป็นมา | 1 |
| 1.2 รูปแบบโปรแกรมภาษาซี | 1 |
| 1.3 ตัวแปร | 5 |
| 1.4 ชนิดข้อมูล | 6 |
| 1.5 การใช้ตัวแปร | 9 |
| 1.6 การรับข้อมูลและแสดงผลข้อมูล | 11 |
| แบบฝึกหัดบทที่ 1 | 17 |
| บทที่ 2. ตัวดำเนินการและนิพจน์ | 19 |
| 2.1 ตัวดำเนินการกำหนดค่า | 19 |
| 2.2 ตัวดำเนินการคณิตศาสตร์ | 21 |
| 2.3 ตัวดำเนินการกำหนดค่าแบบผสม | 23 |
| 2.4 ตัวดำเนินการเพิ่มค่าและลดค่า | 24 |
| 2.5 ตัวดำเนินการเปลี่ยนชนิดข้อมูล | 25 |
| 2.6 ตัวดำเนินการคอมมา | 26 |
| 2.7 ตัวดำเนินการความสัมพันธ์ | 27 |
| 2.8 ตัวดำเนินการความเท่ากัน | 28 |
| 2.9 ตัวดำเนินการตรรกะ | 29 |
| 2.10 ตัวดำเนินการเงื่อนไข | 31 |
| แบบฝึกหัดบทที่ 2 | 33 |
| บทที่ 3. คำสั่งควบคุม | 35 |
| 3.1 คำสั่ง if | 35 |
| 3.2 คำสั่ง switch | 44 |
| 3.3 คำสั่ง for | 48 |
| 3.4 คำสั่ง while | 53 |
| 3.5 คำสั่ง do-while | 61 |
| แบบฝึกหัดบทที่ 3 | 64 |
| บทที่ 4. ฟังก์ชัน | 69 |
| 4.1 แนวความคิดในการออกแบบฟังก์ชัน | 69 |
| 4.2 รูปแบบของฟังก์ชัน | 73 |
| 4.3 การประกาศโปรโตไทป์ของฟังก์ชัน | 77 |

| | | |
|----------|-------------------------------------------|-----|
| 4.4 | ขอบเขต | 79 |
| 4.5 | ฟังก์ชันแบบเรียกซ้ำ | 81 |
| 4.6 | ตัวอย่างเพิ่มเติม | 84 |
| 4.7 | ไลบรารีมาตรฐานของภาษาซี | 87 |
| | แบบฝึกหัดบทที่ 4 | 90 |
| บทที่ 5. | พอยน์เตอร์ | 94 |
| 5.1 | พอยน์เตอร์กับแอดเดรส | 94 |
| 5.2 | การประกาศตัวแปรพอยน์เตอร์ | 95 |
| 5.3 | การกำหนดค่าและการอ่านค่าตัวแปรพอยน์เตอร์ | 95 |
| 5.4 | พอยน์เตอร์และอาร์กิวเมนต์ของฟังก์ชัน | 99 |
| | แบบฝึกหัดบทที่ 5 | 110 |
| บทที่ 6. | ตัวแปรชุด | 113 |
| 6.1 | รูปแบบการประกาศตัวแปรชุด | 113 |
| 6.2 | การใช้พอยน์เตอร์กับตัวแปรชุด | 125 |
| 6.3 | ตัวแปรชุดของตัวอักษร | 128 |
| 6.4 | การคำนวณกับแอดเดรส | 131 |
| 6.5 | ฟังก์ชันมาตรฐานของสตริง | 133 |
| 6.6 | ตัวแปรชุดแบบหลายมิติ | 135 |
| 6.7 | การกำหนดค่าเริ่มต้นให้กับตัวแปรชุด 2 มิติ | 138 |
| 6.8 | การใช้งานตัวแปรชุด 2 มิติ | 138 |
| 6.9 | คอมมานไลน์อาร์กิวเมนต์ | 146 |
| | แบบฝึกหัดบทที่ 6 | 149 |
| บทที่ 7. | โครงสร้างและยูเนียน | 152 |
| 7.1 | ความรู้ทั่วไปเกี่ยวกับโครงสร้าง | 152 |
| 7.2 | การใช้งานตัวแปรโครงสร้าง | 153 |
| 7.3 | การเก็บข้อมูลแบบโครงสร้าง | 158 |
| 7.4 | การใช้ข้อมูลแบบโครงสร้างกับฟังก์ชัน | 160 |
| 7.5 | การใช้พอยน์เตอร์กับตัวแปรโครงสร้าง | 163 |
| 7.6 | การใช้คำสั่ง typedef กับโครงสร้าง | 167 |
| 7.7 | การใช้ตัวแปรชุดเป็นสมาชิกของโครงสร้าง | 169 |
| 7.8 | ตัวแปรชุดของโครงสร้าง | 172 |
| 7.9 | ยูเนียน | 178 |
| | แบบฝึกหัดท้ายบทที่ 7 | 182 |

| | |
|------------------------------------------------|-----|
| บทที่ 8. การจัดการเพิ่มข้อมูล | 185 |
| 8.1 ฟังก์ชันที่ใช้สำหรับการประมวลผลเพิ่มข้อมูล | 185 |
| 8.2 การบันทึกข้อมูลลงเพิ่มข้อมูล | 187 |
| 8.3 การอ่านข้อมูลจากเพิ่มข้อมูล | 188 |
| 8.4 เพิ่มข้อมูลแบบเข้าถึงโดยตรง | 191 |
| 8.5 เพิ่มข้อมูลแบบเรคคอร์ด | 192 |
| 8.6 อุปกรณ์มาตรฐานในการนำเข้าและแสดงผลข้อมูล | 195 |
| 8.7 การรองรับความผิดพลาดจากการทำงาน | 196 |
| 8.8 ตัวอย่างเพิ่มเติม | 197 |
| แบบฝึกหัดท้ายบทที่ 8 | 204 |
| บรรณานุกรม | 205 |

Introduction to C Programming Language

ภาษาโปรแกรม (Programming Languages) ที่มีการคิดค้นขึ้นมาใช้กับคอมพิวเตอร์นั้นมีหลายพันภาษา แต่ภาษาที่เป็นที่รู้จักและเป็นที่ยอมรับใช้ทั่วไปนั้นอาจจะมีเพียงหลายสิบภาษา เช่น โคบอล (COBOL) ปาสคาล (Pascal) เดลไฟล์ (Delphi) วิวอลเบสิก (Visual Basic) ซี (C) จาวา (Java) เป็นต้น ซึ่งแต่ละภาษาสร้างขึ้นด้วยวัตถุประสงค์ที่ต่างกันและมีจุดเด่นของภาษาที่ต่างกัน ภาษาซี (C Programming Language) เป็นภาษาเชิงโครงสร้างที่มีการออกแบบโปรแกรมในลักษณะโมดูลที่มีจุดเด่นในเรื่องของประสิทธิภาพการทำงานที่เร็ว มีความยืดหยุ่นในการเขียนโปรแกรมสูง

เนื่องจากมีผู้ผลิตคอมพิวเตอร์เพื่อใช้แปลภาษาซีหลายบริษัท ตัวอย่างต่าง ๆ ที่นำเสนอในหนังสือเล่มนี้เป็นตัวอย่างที่นำเสนอโดยใช้คอมพิวเตอร์ของ Turbo C เวอร์ชัน 3.0 ของบริษัทบอร์แลนด์ โดยพยายามเขียนในรูปแบบที่เป็นมาตรฐานหากผู้อ่านนำไปใช้กับคอมพิวเตอร์ของบริษัทอื่นจะได้มีการปรับแก้ไม่มากนัก เพื่อให้ผู้อ่านได้เห็นภาพการพัฒนาโปรแกรมเชิงโครงสร้างอย่างชัดเจน

1. ประวัติความเป็นมา

ภาษาซีได้รับการพัฒนาขึ้นโดยเดนนิส ริทชี (Dennis Ritchie) ขณะทำงานอยู่ที่เบลล์แล็บอราทอรี (Bell Laboratories) โดยพัฒนาขึ้นจากหลักการพื้นฐานของภาษาบี (B) และบีซีพีแอล (BCPL) ในช่วงปี ค.ศ. 1971 ถึง 1973 แต่ได้เพิ่มชนิดข้อมูลและความสามารถอื่น ๆ ให้มากขึ้น และนำภาษาซีไปใช้พัฒนาระบบปฏิบัติการยูนิกซ์ (UNIX) บนเครื่องคอมพิวเตอร์ DEC PDP-11 ภาษาซีเป็นที่นิยมใช้อย่างแพร่หลายในช่วงต้นทศวรรษที่ 1980 จนกระทั่งมีความพยายามกำหนดมาตรฐานของภาษาเพื่อให้สามารถใช้ภาษาซีได้บนเครื่องคอมพิวเตอร์ใด ๆ ในปี ค.ศ. 1983 โดย ANSI (The American National Standards Institute) ได้ตั้งคณะกรรมการ X3J11 เพื่อร่างมาตรฐานดังกล่าว และได้รับการตรวจสอบและยอมรับโดย ANSI และ ISO (The International Standards Organization) โดยมีการตีพิมพ์มาตรฐานของภาษาซีในปี ค.ศ. 1990 จากความมีประสิทธิภาพและสามารถทำงานบนเครื่องคอมพิวเตอร์ใด ๆ ของภาษาซีจึงได้มีการนำภาษาซีไปใช้ในการพัฒนาระบบปฏิบัติการต่าง ๆ และใช้เป็นต้นแบบของภาษาอื่น ๆ ที่สำคัญในปัจจุบัน เช่น ซีพลัสพลัส (C++) จาวา (Java) เป็นต้น

2. รูปแบบโปรแกรมภาษาซี

ในการเขียนภาษาโปรแกรม ผู้เขียนโปรแกรมจะต้องศึกษารูปแบบพื้นฐานของภาษา และไวยากรณ์ของภาษานั้น รูปแบบพื้นฐานของภาษาจะเขียนโปรแกรมในลักษณะของโมดูลคือมีการแบ่งออกเป็นส่วนย่อย ๆ ที่เรียกว่า ฟังก์ชัน (Function) แสดงดังตัวอย่างที่ 1.1 และรูปที่ 1.1

ตัวอย่างที่ 1.1 แสดงตัวอย่างโปรแกรมภาษาซีเบื้องต้น

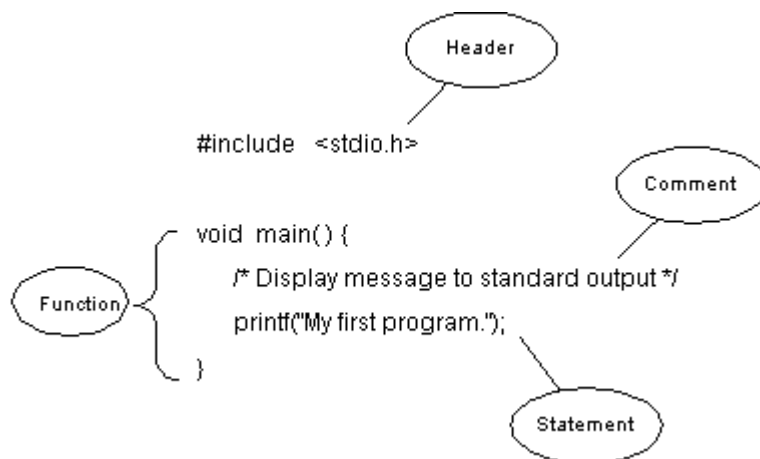
```
#include <stdio.h>

void main() {
    /* Display message to standard output */
    printf("My first program.");
}
```

ผลการทำงานของโปรแกรม

My first program.

ระวัง - การพิมพ์ตัวอักษรตัวพิมพ์ใหญ่และตัวพิมพ์เล็กต่างกัน จะทำให้เกิดความผิดพลาด
- ตรวจสอบว่ามีการพิมพ์ข้อความต่าง ๆ เหมือนกับตัวอย่าง



รูปที่ 1.1 แสดงส่วนประกอบของโปรแกรมภาษาซีเบื้องต้น

ส่วนประกอบที่ 1 ส่วนหัว (Header) จะเป็นส่วนที่อยู่ที่ตอนต้นของโปรแกรม โดยอยู่นอกส่วนที่เรียกว่าฟังก์ชัน ที่ส่วนหัวของโปรแกรมจะประกอบด้วยคำสั่งที่เป็นกรกำหนดค่าหรือกำหนดตัวแปรต่าง ๆ คำสั่งที่ขึ้นต้นด้วยสัญลักษณ์ # เป็นคำสั่งที่เรียกว่า ตัวประมวลผลก่อน (Preprocessor) คือคำสั่งที่จะได้รับการทำก่อนที่จะมีการคอมไพล์โปรแกรม ตัวประมวลผลก่อน ที่สำคัญของภาษาซีแบ่งออกเป็น 2 ประเภทดังนี้

- # include

ในภาษาซีจะมีฟังก์ชันมาตรฐานที่ผู้ผลิตคอมไพเลอร์ได้จัดเตรียมไว้ให้ ซึ่งมักจะเกี่ยวข้องกับการรับข้อมูล การแสดงผลข้อมูล การคำนวณ และอื่น ๆ ซึ่งผู้เขียนโปรแกรมสามารถเรียกใช้งานได้ทันที โดยไม่ต้องเขียนโปรแกรมเอง ในตัวอย่างจะมีการใช้คำสั่ง

printf () ซึ่งเป็นคำสั่งที่ใช้แสดงข้อความออกทางอุปกรณ์แสดงผลมาตรฐาน เช่น จอภาพ คำสั่ง printf () เป็นการเรียกใช้ฟังก์ชันมาตรฐานซึ่งอยู่ในกลุ่มที่เรียกว่า Standard Input and Output เมื่อจะเรียกใช้ฟังก์ชันใดในกลุ่มดังกล่าว จะต้องบอกให้คอมไพเลอร์ไปอ่านค่าที่อยู่ในอินคลูซัฟไฟล์ที่ชื่อ stdio.h มาไว้ที่ส่วนต้นของโปรแกรม โดยใช้คำสั่ง

```
#include <stdio.h>
```

เพราะฉะนั้นผู้เขียนโปรแกรมควรจะศึกษาฟังก์ชันมาตรฐานที่คอมไพเลอร์แต่ละบริษัทได้เตรียมไว้ให้ว่าคำสั่งใดใช้คู่กับอินคลูซัฟไฟล์ใด

- **# define**

ใช้สำหรับการกำหนดค่าคงที่ ตัวอย่างเช่น

```
#define YES 1
```

คำสั่งดังกล่าวเป็นการกำหนดว่า หากที่ใดในโปรแกรมมีคำว่า YES จะถูกแทนที่ด้วยค่าทางขวามือ ในที่นี้คือ 1

นอกจากในส่วนหัวของโปรแกรมอาจจะมีการประกาศตัวแปร และส่วนของการประกาศโปรโตไทป์ไว้ที่ส่วนหัวของโปรแกรมได้อีกด้วย ซึ่งจะกล่าวถึงในบทต่อ ๆ ไป

ส่วนประกอบที่ 2 ฟังก์ชัน (Function) ส่วนของฟังก์ชันคือส่วนของคำสั่งที่บอกให้คอมพิวเตอร์ทำงานต่าง ๆ เช่น การรับข้อมูล การคำนวณ การแสดงผล เป็นต้น โปรแกรมภาษาซีจะประกอบด้วยฟังก์ชันย่อยหลาย ๆ ฟังก์ชัน แต่จะมีฟังก์ชันหลักฟังก์ชันหนึ่งซึ่งชื่อว่าฟังก์ชัน main () เสมอ โดยที่การทำงานของโปรแกรมจะต้องเริ่มการทำงานจากฟังก์ชันนี้

กฎพื้นฐานที่สำคัญในภาษาซี

- การพิมพ์ตัวอักษรตัวพิมพ์ใหญ่และตัวพิมพ์เล็กในภาษาซีนั้นในผลลัพธ์ที่แตกต่างกัน (Case Sensitive) ตัวอย่างเช่น หากมีการพิมพ์ main () กลายเป็น Main () ก็จะทำให้เกิดความผิดพลาดขึ้น
- ฟังก์ชันของภาษาซีจะแบ่งขอบเขตของฟังก์ชันแต่ละฟังก์ชันด้วยเครื่องหมาย { } ในตัวอย่างมีฟังก์ชัน void main () คำว่า void จะบอกให้รู้ว่าเมื่อฟังก์ชันนี้ทำงานเสร็จจะไม่มีค่ากลับไปยังสิ่งที่เรียกใช้งานฟังก์ชัน ในกรณีของฟังก์ชัน main () ก็คือจะไม่มีค่าใด ๆ กลับไปยังระบบปฏิบัติการ หลังฟังก์ชันจะต้องตามด้วย () เสมอ โดยที่ภายในวงเล็บจะประกอบด้วยคำสั่งเข้ามายังฟังก์ชัน ที่เรียกว่าพารามิเตอร์ (Parameter) หรืออาจจะไม่มีค่าใด ๆ ส่งเข้ามาก็ได้
- คำสั่งต่าง ๆ ซึ่งต้องเขียนอยู่ในฟังก์ชันเสมอ แบ่งเป็น 2 ส่วนคือส่วนของการประกาศตัวแปรที่ต้องการใช้ในฟังก์ชัน และส่วนของคำสั่งเพื่อทำงานใดงานหนึ่ง ในที่นี้มีเฉพาะคำสั่งที่ใช้ในการ

แสดงผลลัพธ์ออกทางจอภาพ คือ printf() ใช้สำหรับการแสดงผลลัพธ์ออกทางจอภาพ หากต้องการแสดงข้อความใด ๆ ออกทางจอภาพให้เขียนข้อความนั้นอยู่ภายในเครื่องหมาย “ “

- คำสั่งในภาษาซีจะต้องปิดท้ายด้วยเครื่องหมาย ; (Semicolon) เนื่องจากภาษาซีจะใช้เครื่องหมาย ; ในการแยกคำสั่งต่าง ๆ ออกจากกัน การเว้นบรรทัดหรือการเขียนคำสั่งไม่ต่อเนื่องกันจะไม่มีผลต่อคอมไพเลอร์ แต่เป็นการช่วยให้ผู้เขียนโปรแกรมอ่านโปรแกรมได้ง่ายขึ้นเท่านั้น

จากตัวอย่างที่ 1.1 อาจเขียนโปรแกรมในลักษณะต่าง ๆ ซึ่งไม่มีผลต่อการทำงานของคอมไพเลอร์ดังตัวอย่างที่ 1.2 และ 1.3

ตัวอย่างที่ 1.2 แสดงตัวอย่างโปรแกรมภาษาซีเบื้องต้น

```
#include <stdio.h> void main() {      /* Display message to standard output */      printf
("My first program."); }
```

ตัวอย่างที่ 1.3 แสดงตัวอย่างโปรแกรมภาษาซีเบื้องต้น

```
#include
<stdio.h>
void
main
(
)
{
/* Display message
to standard
output */
printf
(
"My first program."
)
;
}
```

ไม่ว่าผู้ใช้จะเขียนโปรแกรมในลักษณะที่ 1.1 1.2 หรือ 1.3 คอมไพเลอร์จะทำงานได้ผลลัพธ์เดียวกันเสมอ เพราะฉะนั้นการเขียนโปรแกรมที่เป็นระเบียบจะช่วยให้ผู้เขียนโปรแกรมหรือผู้อื่นที่มาอ่านโปรแกรมอ่านได้ง่ายขึ้น โดยทั่วไปมักใช้ระบบของการเยื้อง (Indentation) เข้ามาช่วยให้โปรแกรมอ่านได้ง่ายขึ้น พิจารณา

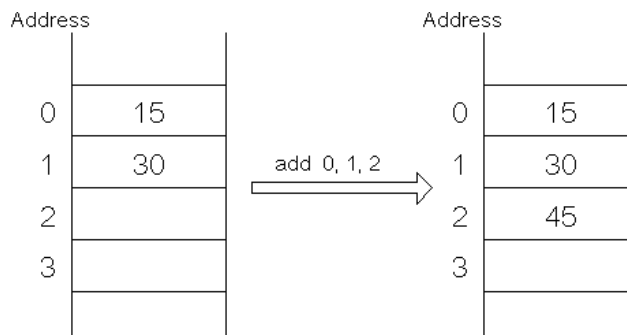
จากตัวอย่างที่ 1.2 และ 1.3 ซึ่งไม่มีการเชื่อมกับตัวอย่างที่ 1.1 ที่เขียนโดยมีลักษณะของการเชื่อม การเชื่อมนั้น อาจจะใช้การเคาะช่องว่าง (Space Bar) เป็นจำนวน 3-5 ทีเพื่อให้เกิดการเชื่อม

นอกจากนี้หากผู้ใช้ต้องการใส่คำอธิบายลงในโปรแกรม ก็สามารถทำได้โดยใช้สิ่งที่เรียกว่า Comment ขอบเขตของ Comment จะขึ้นต้นตั้งแต่ /* จนกระทั่งถึง */ ข้อความใด ๆ ที่อยู่ในขอบเขตของเครื่องหมายดังกล่าวจะไม่ถูกแปลโดยคอมไพเลอร์ โดยทั่วไปจะมีการใช้ Comment เพื่ออธิบายว่าโปรแกรมนั้นทำงานอะไร ใช้อธิบายคำสั่งแต่ละคำสั่ง ใช้อธิบายฟังก์ชันแต่ละฟังก์ชัน และใช้อธิบายกลุ่มของคำสั่ง โดยเขียน Comment ไว้ด้านบนหรือด้านข้างของสิ่งที่ต้องการอธิบาย เช่น หากเป็นการอธิบายโปรแกรมจะเขียน Comment ไว้ที่ต้นของโปรแกรมนั้น หากเขียนอธิบายฟังก์ชันจะเขียน Comment ไว้ด้านบนของฟังก์ชันที่ต้องการอธิบาย เขียนอธิบายคำสั่งอาจจะเขียน Comment ไว้ด้านบนหรือด้านข้างของคำสั่งนั้น ๆ และเขียนอธิบายกลุ่มคำสั่งก็จะเขียนอธิบายไว้ด้านของกลุ่มคำสั่งนั้น

3. ตัวแปร

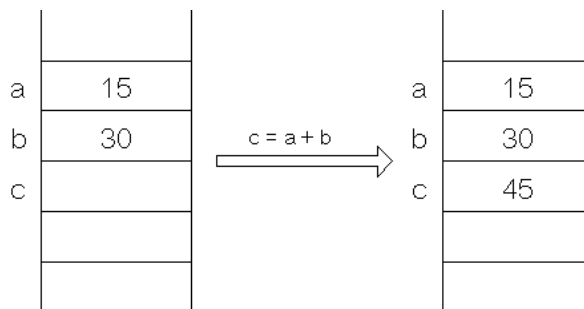
ในการเขียนโปรแกรมคอมพิวเตอร์กระบวนการสำคัญที่เกิดขึ้น คือ การรับข้อมูล การประมวลผล ข้อมูล และการแสดงผลข้อมูล จะเห็นว่าสิ่งที่เป็นส่วนสำคัญที่สุดคือข้อมูล การทำงานของโปรแกรมขณะใดขณะหนึ่งจะต้องมีการเก็บข้อมูลไว้ในคอมพิวเตอร์ โดยรับข้อมูลจากอุปกรณ์รับข้อมูลไปเก็บไว้ในส่วนที่เรียกว่าหน่วยความจำ และส่งข้อมูลจากหน่วยความจำไปประมวลผลในหน่วยประมวลผลกลาง โดยผ่านคำสั่งต่าง ๆ เมื่อประมวลผลเสร็จแล้วก็นำผลลัพธ์ที่ได้กลับมาเก็บไว้ในหน่วยความจำอีก เมื่อต้องการให้แสดงผลก็จะใช้คำสั่งให้ไปอ่านข้อมูลจากหน่วยความจำส่งข้อมูลนั้นไปยังอุปกรณ์แสดงผล

พิจารณาจากตัวอย่างการบวกเลข แสดงดังรูปที่ 1.2 โดยให้รับค่าจำนวนเต็ม 2 ค่าจากผู้ใช้ และนำไปเก็บอยู่ในหน่วยความจำตำแหน่งที่ 0 มีค่า 15 ตำแหน่งที่ 1 มีค่า 30 หากต้องการบวกนำค่าทั้ง 2 มาบวกกัน และเก็บไว้ในตำแหน่งที่ 2 จะต้องใช้คำสั่งในการบวก สมมติดังตัวอย่าง Add 0, 1, 2 คือการเอาค่า ณ ตำแหน่งที่ 0 และ 1 มาบวกกัน และนำไปเก็บที่ตำแหน่งที่ 2 จะได้ว่ามีการอ่านค่า 15 จากตำแหน่งที่ 0 ไปบวกกับค่า 20 จากตำแหน่งที่ 1 ได้ผลลัพธ์คือ 45 นำไปเก็บในตำแหน่งที่ 2



รูปที่ 1.2 แสดงแบบจำลองการบวกเลขในคอมพิวเตอร์

ในความเป็นจริงตำแหน่งต่าง ๆ ไม่ได้อ้างด้วยเลข 0 1 หรือ 2 แต่อ้างด้วยระบบการอ้างที่อยู่ซึ่งเป็น
 ชั้นซ้อนกว่านั้น การเขียนในลักษณะที่เห็นในตัวอย่างเป็นการเขียนในภาษาคอมพิวเตอร์ยุคแรก เช่น
 แอสเซมบลี (Assembly) เพื่อความสะดวกในการอ้างถึงข้อมูล ณ ตำแหน่งต่าง ๆ ในหน่วยความจำ จึงได้มีการ
 พัฒนากันต่อมา จนกระทั่งเป็นการอ้างในระบบของชื่อตัวแปร (Variable) ที่ใช้ในปัจจุบัน พิจารณาจากตัว
 อย่างที่ใช้ระบบชื่อตัวแปรแสดงดังรูปที่ 1.3 ผู้เขียนโปรแกรมสร้างตัวแปรขึ้นมา จะเกิดการจองพื้นที่ในหน่วย
 ความจำให้โดยอัตโนมัติ ผู้ใช้ไม่จำเป็นต้องรู้ว่าตัวแปรนั้นอยู่ที่ตำแหน่งใด การอ้างถึงชื่อตัวแปรคือการอ้างถึง
 ข้อมูลในหน่วยความจำ เพราะฉะนั้นหากผู้ใช้สร้างตัวแปร a และ b เพื่อเก็บข้อมูล 15 และ 20 ตามลำดับ เมื่อ
 ต้องการหาผลบวกและนำค่าไปเก็บไว้ที่หนึ่ง ก็สร้างตัวแปรขึ้นมาเพื่อเก็บผลลัพธ์สมมติชื่อว่า c การสั่งให้
 ทำงานก็สามารถทำได้โดยใช้คำสั่ง $c = a + b$ ซึ่งอ่านได้เข้าใจได้ง่ายกว่าคำสั่ง `add 0, 1, 2`



รูปที่ 1.3 แสดงแบบจำลองการบวกเลขโดยอ้างชื่อตัวแปร

4. ชนิดข้อมูล

ในการใช้งานตัวแปรนั้นสิ่งสำคัญที่จะต้องคำนึงถึงคือข้อมูลที่เก็บอยู่ภายในตัวแปรนั้น ข้อมูลแต่ละ
 ชนิดมีคุณสมบัติแตกต่างกันไป เช่น เป็นเลขจำนวนเต็ม เป็นเลขจำนวนจริง เป็นตัวอักษร เป็นต้น ผู้เขียน
 โปรแกรมจะต้องกำหนดชนิดข้อมูลให้กับตัวแปรโดยสอดคล้องกับข้อมูลที่ต้องการเก็บ ภาษาซีแบ่งข้อมูลออก
 เป็นชนิดต่าง ๆ ซึ่งมีขนาดพื้นที่ที่ต้องใช้เก็บข้อมูลในหน่วยความจำแตกต่างกันขึ้นอยู่กับชนิดข้อมูลและบริษัทผู้
 ผลิตคอมพิวเตอร์ ซึ่งสามารถดูรายละเอียดได้จากอินคลูซัฟไฟล์ `limits.h` และ `float.h` แสดงดังตาราง 1.1

ตาราง 1.1 แสดงคุณสมบัติของชนิดข้อมูลพื้นฐาน

| ชนิดข้อมูล | ขนาด (ไบต์) | ค่าที่เก็บ |
|------------|------------------|------------------------------------------------|
| char | 1 | ตัวอักษร ASCII 1 ตัว ตั้งแต่ 0 ถึง 255 |
| int | 2 | ค่าจำนวนเต็มตั้งแต่ 32767 ถึง -32768 |
| short | 2 | ค่าจำนวนเต็มตั้งแต่ 32767 ถึง -32768 |
| long | 4 | ค่าจำนวนเต็มตั้งแต่ 2147483647 ถึง -2147483648 |
| unsigned | unsigned int 2 | ค่าจำนวนเต็มตั้งแต่ 0 ถึง 65535 |
| | unsigned short 2 | ค่าจำนวนเต็มตั้งแต่ 0 ถึง 65535 |
| | unsigned long 4 | ค่าจำนวนเต็มตั้งแต่ 0 ถึง 4294967295 |

ตาราง 1.1 (ต่อ) แสดงคุณสมบัติของชนิดข้อมูลพื้นฐาน

| ชนิดข้อมูล | ขนาด (ไบต์) | ค่าที่เก็บ |
|------------|-------------|----------------------------------------------------------------------|
| float | 4 | ค่าจำนวนจริงตั้งแต่ 3.4×10^{-38} ถึง 3.4×10^{38} |
| double | 8 | ค่าจำนวนจริงตั้งแต่ 1.7×10^{-308} ถึง 1.7×10^{308} |

จากชนิดของข้อมูลดังกล่าว เมื่อนำมาจัดเป็นประเภทข้อมูลเราสามารถแบ่งชนิดของข้อมูลในภาษาซีออกเป็นประเภทหลักได้ 3 ชนิด คือ

1. จำนวนเต็ม ได้แก่ข้อมูลชนิด int, short, long, unsigned int, unsigned short และ unsigned long
2. จำนวนจริง คือ ข้อมูลที่เป็นเลขจำนวนจริง มีทศนิยม หรืออยู่ในรูปของนิพจน์วิทยาศาสตร์ ได้แก่ข้อมูลชนิด float และ double
3. ตัวอักษรและสตริง (String) ได้แก่ข้อมูลชนิด char ซึ่งเก็บข้อมูลได้ 1 อักขระ และข้อมูลที่เป็นชุดของข้อมูล char (Array of char) ที่ใช้เก็บสตริงซึ่งจะกล่าวถึงต่อไป

● รูปแบบการเขียนค่าต่าง ๆ ที่เป็นไปได้ของข้อมูลแต่ละชนิด ได้แก่

- จำนวนเต็ม กรณีเขียนเป็นเลขฐาน 10 เช่น 125 0 -154

กรณีเขียนเป็นเลขฐาน 8 ให้ขึ้นต้นด้วย 0 เช่น 0377 0177 0000 01

กรณีเขียนเป็นเลขฐาน 16 ให้ขึ้นต้นด้วย 0x หรือ 0X เช่น 0xFF 0x14 0x4B

กรณีข้อมูลเป็นชนิด long ให้ลงท้ายด้วย l หรือ L เช่น 123L 0437l 0x4FFL

- จำนวนจริง เขียนรูปแบบทศนิยม เช่น 12.524 0.1259 0.00001

เขียนรูปแบบนิพจน์วิทยาศาสตร์ เช่น 1.0E4 12.432e-4 1E-4 2E+3

กรณีข้อมูลเป็น float เช่น 1.4321F 472.324f

กรณีข้อมูลเป็น double เช่น 232.98D 1.2323d

- ตัวอักษร เช่น 'a' 'W' '\a' '\t' โดยที่อักขระใดขึ้นต้นด้วย \

แสดงว่าเป็นอักขระพิเศษ เช่น '\a' แทนการส่งเสียง

'\t' แทนการย่อหน้า

'\n' แทนการขึ้นบรรทัดใหม่

'\0' แทนค่า NULL ซึ่งหมายถึงค่าว่าง

นอกจากนี้ยังสามารถใช้ในลักษณะเลขฐาน 8 และเลขฐาน 16 ได้ดังนี้ ตัวอย่างเช่น 'X' ในตาราง ASCII ตรงกับเลขฐาน 8 คือ 130 และเลขฐาน 16 คือ 58 เขียนแทนค่าคงที่อักขระได้คือ '\130' และ '\x58' ตามลำดับ

```
char ch1='X';
```

```
char ch2 = '\130';
```

```
char ch3 = '\x58';
```

ถ้าสั่งให้พิมพ์ค่าตัวแปร ch1 ch2 และ ch3 จะพิมพ์ค่าเดียวกันคือ X

ตัวอย่างที่ 1.4 แสดงตัวอย่างการใช้ค่าของตัวแปรชนิด char

```
#include <stdio.h>

void main() {
    int no;
    char ch;
    ch = 'J';
    printf("char : %c, dec : %d, oct : %o, hex : %x", ch, ch, ch, ch);
    no = ch;
    printf("\nno : %d, ch : %c", no, ch);
    no = 68;
    ch = no;
    printf("\nno : %d, ch : %c", no, ch);
}
```

ผลการทำงานของโปรแกรม

```
char : J, dec : 74, oct : 112, hex : 4a
no : 74, ch : J
no : 68, ch : D
```

-สตริง จะเขียนอยู่ในเครื่องหมาย " เช่น "Hello" โดยที่ในสตริงทุกตัวจะมีอักขระพิเศษที่ไม่สามารถมองเห็นคือ '\0' หรือ NUL เป็นตัวสุดท้ายเสมอ ในขณะที่อักขระจะจองพื้นที่หน่วยความจำเพียง 1 ไบต์ ในตัวอย่างจะได้ว่าเกิดการจองพื้นที่ขึ้น 6 ตัวอักษร ตัวอย่างการประกาศตัวแปรสตริง

```
char msg[] = "Hello";
```

แสดงการจองพื้นที่ในหน่วยความจำให้กับตัวแปร msg ดังรูปที่ 1.4

| | | | | | |
|---|---|---|---|---|----|
| H | e | l | l | o | \0 |
|---|---|---|---|---|----|

รูปที่ 1.4 แสดงการจองพื้นที่หน่วยความจำของตัวแปร msg

5. การใช้ตัวแปร

เมื่อต้องการใช้ตัวแปร จะต้องมีการประกาศชื่อตัวแปรที่ต้องการใช้งานนั้น มีรูปแบบคือ

```
ประเภทข้อมูล ชื่อตัวแปร ;
```

ตัวอย่างของการประกาศตัวแปร เช่น

```
float score;
```

```
int age;
```

```
char ch;
```

```
float width, height, length;
```

กรณีที่มีตัวแปรมากกว่า 1 ตัวที่มีชนิดเดียวกัน สามารถประกาศไว้ในคำสั่งเดียวกันได้โดยใช้เครื่องหมาย , คั่นระหว่างตัวแปรแต่ละตัว

กฎการตั้งชื่อ

ในภาษาซีมีการกำหนดกฎในการตั้งชื่อ Identifier ต่าง ๆ อันได้แก่ ชื่อตัวแปร ชื่อฟังก์ชัน ชื่อค่าคงที่
ดังนี้

- ให้ใช้ตัวอักษร a ถึง z A ถึง Z เลข 0 ถึง 9 และ _ (Underscore) ประกอบกันเป็นชื่อ
- ขึ้นต้นด้วยตัวอักษรหรือ _
- ตัวอักษรตัวพิมพ์ใหญ่ ตัวพิมพ์เล็กมีผลต่อการตั้งชื่อและการเรียกใช้งาน
- ชื่อนั้นจะต้องไม่ซ้ำกับคำหลัก (Keyword) ซึ่งภาษาซีจองไว้ใช้ คือ

| | | | |
|----------|--------|----------|----------|
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |
| char | extern | return | union |
| const | float | short | unsigned |
| continue | for | signed | void |
| default | goto | sizeof | volatile |
| do | if | static | while |

คำแนะนำในการตั้งชื่อ

ในการเขียนโปรแกรมที่ดีนั้นเราควรทำการตั้งชื่อของตัวแปร ค่าคงที่ ฟังก์ชัน ให้อยู่ในรูปแบบมาตรฐานดังนี้

- ให้ตั้งชื่อที่สื่อความหมายบอกให้รู้ว่าตัวแปรนั้นใช้ทำอะไร
- ขึ้นต้นด้วยตัวอักษร
- กรณีตั้งชื่อตัวแปรมักจะหรือฟังก์ชันมักจะใช้ตัวอักษรตัวพิมพ์เล็ก
- ค่าคงที่ที่กำหนดโดย #define มักจะใช้ตัวอักษรตัวพิมพ์ใหญ่ทั้งหมด
- กรณีที่ชื่อตัวแปรประกอบด้วยคำหลาย ๆ คำ อาจจะใช้ตัวอักษรตัวพิมพ์ใหญ่ขึ้นต้นคำในลำดับต่อมา หรือใช้ _ แยกระหว่างคำ เช่น totalScore หรือ total_score

ตัวอย่างการตั้งชื่อตัวแปร เช่น

```
totalscore = score1 + score2 + score3;
```

ย่อมจะทำความเข้าใจได้ง่ายกว่า

```
robert = willy + bird + smith;
```

ตัวอย่างชื่อตัวแปรที่ใช้ได้ ได้แก่

| | | | |
|--------|-----------|----------|---------------|
| salary | _MAX | MONEY | Average_score |
| x | totalPage | recordno | annual2001 |

ตัวอย่างของการใช้คำสั่งประกาศตัวแปรไม่ถูกต้องและไม่เหมาะสมได้แก่

| | |
|-----------------------|-------------------------------------------|
| float \$dollar; | ใช้เครื่องหมาย \$ ในชื่อตัวแปรไม่ได้ |
| int my age; | ชื่อตัวแปรห้ามเว้นวรรค |
| integer age; | ประเภทข้อมูล integer ไม่มีในภาษาซี |
| int a. b. c; | ใช้เครื่องหมาย . แทนที่จะใช้เครื่องหมาย , |
| char ch: | ใช้เครื่องหมาย : แทนที่จะใช้เครื่องหมาย ; |
| float student-height; | ใช้เครื่องหมาย - แทนที่จะใช้เครื่องหมาย _ |
| float 1Score; | ชื่อตัวแปรห้ามขึ้นต้นด้วยตัวเลข |
| float score; | } ห้ามประกาศชื่อซ้ำกัน |
| int score; | |

เมื่อมีการประกาศตัวแปร จะเกิดกระบวนการจองพื้นที่ในหน่วยความจำให้กับตัวแปรตัวนั้นมีขนาดเท่ากับชนิดของข้อมูลที่กำหนด เมื่อใดที่มีการอ้างถึงชื่อตัวแปรก็จะเป็นการอ้างถึงค่าที่เก็บอยู่ในพื้นที่หน่วยความจำนั้น สิ่งที่ต้องระวังคือ ควรจะมีการกำหนดค่าเริ่มต้นให้กับตัวแปรนั้น ๆ เสมอ เพราะพื้นที่ในหน่วยความจำที่ถูกจองนั้นอาจจะมีค่าบางอย่างอยู่ภายใน ตัวอย่างของการกำหนดค่าเริ่มต้นให้กับตัวแปร คือ

```
int sum=0;
float height=0.0f;
```

จากคำสั่งข้างต้นระบบจะทำการจองพื้นที่ในหน่วยความจำให้กับตัวแปรชื่อ sum มีขนาดเท่ากับ int (2 ไบต์) และกำหนดให้มีค่าเริ่มต้นเท่ากับ 0 เมื่อมีการอ้างถึงชื่อตัวแปรก็จะได้ค่าคือ 0 และจองพื้นที่ในหน่วยความจำให้กับตัวแปร height มีขนาดเท่ากับ float (4 ไบต์) และกำหนดให้มีค่าเริ่มต้นเท่ากับ 0.0

6. การรับข้อมูลและแสดงผลข้อมูล

ภาษาซีได้เตรียมฟังก์ชันมาตรฐานที่ใช้ในการรับและแสดงผลข้อมูลให้กับผู้พัฒนาโปรแกรมหลายฟังก์ชัน แต่ในที่นี้จะกล่าวถึงเฉพาะฟังก์ชันในการรับข้อมูล คือ ฟังก์ชัน scanf() และฟังก์ชันในการแสดงผลข้อมูลคือฟังก์ชัน printf() ในส่วนของการใช้งานพื้นฐาน ซึ่งก่อนจะใช้งานฟังก์ชันดังกล่าวที่ส่วนหัวของโปรแกรมจะต้องมีคำสั่ง

```
#include <stdio.h>
```

6.1 การรับข้อมูล

ฟังก์ชันที่ใช้ในการรับข้อมูลมีรูปแบบของการใช้งานคือ

```
scanf(รูปแบบ, อาร์กิวเมนต์1, อาร์กิวเมนต์2, ...);
```

ในการรับข้อมูลผู้เขียนโปรแกรมจะต้องกำหนดรูปแบบของข้อมูลที่ต้องการรับซึ่งสอดคล้องกับชนิดของข้อมูลที่ต้องการรับเข้า โดยที่ผู้ใช้ต้องส่งตำแหน่ง (หรือแอดเดรสในหน่วยความจำ - Address) ของตัวแปรที่ต้องการรับเข้าไปยังฟังก์ชัน โดยระบุในตำแหน่งของอาร์กิวเมนต์

ตัวอย่างเช่น ต้องการรับข้อมูลเดือนและปีเป็นจำนวนเต็ม จะต้องใช้คำสั่ง

```
int month, year;
scanf("%d %d", &month, &year);
```

จากตัวอย่างรูปแบบ คือ “%d %d” บอกให้รู้ว่าต้องรับข้อมูล 2 ตัว โดยที่ข้อมูลนั้นแยกจากกันด้วยช่องว่าง %d ตัวแรกแทนข้อมูลตัวแรกที่ป้อนเข้าสู่ระบบจะนำไปเก็บที่ตัวแปร month %d ตัวที่ 2 แทนข้อมูลตัวที่ 2 ที่ป้อนเข้าสู่ระบบจะถูกนำไปเก็บที่ตัวแปร year ในที่นี้จะระบุตำแหน่งของตัวแปรในหน่วยความจำด้วยเครื่องหมาย & (Address Operator)

รูปแบบของการรับข้อมูลในภาษาซี จะขึ้นต้นด้วยสัญลักษณ์ % อยู่ในเครื่องหมาย “ ” รูปแบบการรับข้อมูลแสดงได้ดังตาราง 1.2

ตาราง 1.2 แสดงรูปแบบของการรับข้อมูล

| รูปแบบ | คำอธิบายการใช้งาน |
|--------|------------------------------------------------------------------|
| d, i | รับข้อมูลจำนวนเต็มและแปลงข้อมูลเป็น int |
| ld | รับข้อมูลจำนวนเต็มและแปลงข้อมูลเป็น long |
| u | รับข้อมูลเป็นจำนวนเต็มบวกและแปลงข้อมูลเป็น unsigned |
| o | รับข้อมูลเป็นจำนวนเต็มบวกของเลขฐาน 8 และแปลงข้อมูลเป็น unsigned |
| x | รับข้อมูลเป็นจำนวนเต็มบวกของเลขฐาน 16 และแปลงข้อมูลเป็น unsigned |
| c | รับข้อมูลเป็นอักขระ 1 ตัว |
| s | รับข้อมูลเป็นสตริง |
| f | รับข้อมูลทศนิยมและแปลงข้อมูลเป็น float |
| lf | รับข้อมูลทศนิยมและแปลงข้อมูลเป็น double |

ตัวอย่างการรับข้อมูลมีดังนี้

```
scanf("%d/%d/%d", &day, &month, &year);
```

ข้อมูลที่จะป้อนเข้าสู่ระบบ

```
20/7/2001
```

นอกจากนี้ยังสามารถกำหนดขนาดของข้อมูล เช่น

```
scanf("%2d%2d%4d", &day, &month, &year);
```

ข้อมูลที่จะป้อนเข้าสู่ระบบคือ

```
20072001
```

หมายเหตุ

ใน Turbo C กรณีที่มีการรับข้อมูลชนิด char ในโปรแกรม หากในโปรแกรมมีการรับชนิดข้อมูลอื่นด้วย จะพบว่าเมื่อป้อนข้อมูลชนิด char เสร็จ จะไม่มีการหยุดรับข้อมูลในลำดับถัดไป เนื่องจากความผิดพลาดของการทำงานของคอมไพเลอร์ เพราะฉะนั้นควรจะใช้ฟังก์ชัน flushall() ซึ่งเป็นฟังก์ชันมาตรฐานใน stdio.h ช่วยเพื่อแก้ปัญหาดังกล่าว โดยใส่ฟังก์ชัน flushall() ไว้หลังคำสั่ง scanf() ทุกคำสั่ง

6.2 การแสดงผลข้อมูล

การแสดงผลข้อมูลสามารถทำได้โดยการเรียกใช้ฟังก์ชัน printf() มีรูปแบบคือ

```
printf ( รูปแบบ , อาริกิวเมนต์1 , อาริกิวเมนต์2 , ... );
```

ผู้เขียนโปรแกรมสามารถส่งข้อความใด ๆ มายังฟังก์ชัน print() หรือส่งตัวแปรมาพิมพ์ค่าโดยส่งผ่าน มาทางอาริกิวเมนต์ ให้สอดคล้องกับรูปแบบที่กำหนด ตัวอย่างเช่น

```
char name[ ] = "Mickey";
int age = 20;
printf("%s is %d years old.", name, age);
```

ผลลัพธ์ที่ได้คือ

Mickey is 20 years old.

โดยที่ %s ตัวแรกจะถูกแทนที่ด้วยค่าของอาริกิวเมนต์ตัวที่ 1 คือ ตัวแปร name และ %d ตัวที่ 2 จะ ถูกแทนที่ด้วยค่าของอาริกิวเมนต์ตัวที่ 2 คือ ตัวแปร age

นอกจากนี้ผู้เขียนยังสามารถใช้ printf() ในการพิมพ์ข้อความใด ๆ ออกทางจอภาพโดยไม่จำเป็นต้องส่ง อาริกิวเมนต์เข้าไปยังฟังก์ชันก็ได้ ตัวอย่างเช่น

```
printf("Good morning.");
```

ผลลัพธ์ที่ได้คือ

Good morning.

ตัวอย่างของการแทนที่ตัวแปรที่เกิดจากการคำนวณ เช่น

```
int x, y;
x = 7;
y = 2;
printf("The sum of %d and %d is %d\n", x, y, x+y);
```

ผลลัพธ์ที่ได้คือ

The sum of 7 and 2 is 9

คำสั่ง `\n` เป็นอักขระพิเศษที่ได้กล่าวถึงแล้วคือ อักขระที่ใช้สั่งให้มีการเว้นบรรทัด (New Line) ในที่นี้ หลังจากพิมพ์ข้อความเสร็จจะมีการเว้นบรรทัดเกิดขึ้น จากตัวอย่างจะเห็นว่า `%d` ตัวที่ 3 เกิดจากการนำค่าของตัวแปร `x` มาบวกกับตัวแปร `y` ได้ผลลัพธ์เท่าใดจึงส่งให้กับ `%d` ตัวที่ 3 แสดงผล

รูปแบบการแสดงผล ในฟังก์ชัน `printf()` แสดงดังตาราง 1.3

ตาราง 1.3 แสดงรูปแบบของการแสดงข้อมูล

| รูปแบบ | คำอธิบายการใช้งาน |
|--------|---------------------------------------------------------------|
| d, i | ข้อมูลจำนวนเต็มและแปลงข้อมูลเป็น int |
| ld | ข้อมูลจำนวนเต็มและแปลงข้อมูลเป็น long |
| u | ข้อมูลเป็นจำนวนเต็มบวกและแปลงข้อมูลเป็น unsigned |
| o | ข้อมูลเป็นจำนวนเต็มบวกของเลขฐาน 8 และแปลงข้อมูลเป็น unsigned |
| x, X | ข้อมูลเป็นจำนวนเต็มบวกของเลขฐาน 16 และแปลงข้อมูลเป็น unsigned |
| c | ข้อมูลเป็นอักขระ 1 ตัว |
| s | ข้อมูลเป็นสตริง |
| f | ข้อมูลทศนิยมและแปลงข้อมูลเป็น float |
| lf | ข้อมูลทศนิยมและแปลงข้อมูลเป็น double |

การจัดรูปแบบการแสดงผล

กรณีของการใช้รูปแบบในฟังก์ชัน `printf()` จะแตกต่างกับฟังก์ชัน `scanf()` ตรงที่ฟังก์ชัน `printf()` ใช้ในการแสดงผลออกทางหน้าจอ เพราะฉะนั้นจึงมีการกำหนดค่าเพื่อให้สามารถจัดผลลัพธ์ให้แสดงในรูปแบบที่ต้องการได้ โดยปกติแล้วข้อมูลทุกตัวที่แสดงจะแสดงผลจากทางด้านขวามือ เช่น

```
printf("The sum of %5d and %5d is %5s\n", a, b, a+b);
```

ผลลัพธ์ที่ได้คือ

```
The sum of 7 and 2 is 9.
```

จะเกิดการเว้นช่องว่าง 5 ช่อง และแสดงผลจากทางด้านขวามือของช่องว่างนั้น

ตัวอย่างอื่นได้แก่

```
float x=43.34, y=2.231;
```

```
printf("Minus %f with %f, answer is %f", x, y, x-y);
```

ผลลัพธ์ที่ได้คือ

```
Minus 43.340000 with 2.231000, answer is 41.109000
```

แต่หากกำหนดว่า

```
printf("Minus %4.2f with %4.2f, answer is %4.2f");
```

ผลลัพธ์ที่ได้คือ

Minus 43.34 with 2.23, answer is 41.11

การจัดรูปแบบเมื่อใช้สัญลักษณ์ - ประกอบในรูปแบบจะทำให้เกิดการบังคับให้การแสดงผลชิดทางด้านซ้ายมือ แสดงดังตัวอย่างที่ 1.5

ตัวอย่างที่ 1.5 แสดงตัวอย่างการจัดรูปแบบของการแสดงผล

```
#include <stdio.h>
#include <conio.h>
void main() {
    clrscr();
    printf("\n[%s]", "Computer");           /* [Computer] */
    printf("\n[%2s]", "Computer");         /* [Computer] */
    printf("\n[%3s]", "Computer");         /* [Com] */
    printf("\n[%10s]", "Computer");        /* [ Computer] */
    printf("\n[%-10s]", "Computer");       /* [Computer ] */
    printf("\n[%-10.3s]", "Computer");     /* [Com      ] */
    printf("\n");
    printf("\n[%x]", 15);                   /* Hexa [f] */
    printf("\n[%o]", 15);                   /* Octal [17] */

    printf("\n");
    printf("\n[%d]", 100);                  /* [100] */
    printf("\n[%2d]", 100);                 /* [100] */
    printf("\n[%10d]", 100);               /* [ 100] */
    printf("\n[%-10d]", 100);              /* [100  ] */
    printf("\n[%-10.2d]", 100);            /* [100  ] */

    printf("\n");
    printf("\n[%f]", 32.5762);              /* [32.576200] */
    printf("\n[%2f]", 32.5762);            /* [32.58] */
    printf("\n[%10.2f]", 32.5762);         /* [ 32.58] */
    printf("\n[%-10.2f]", 32.5762);       /* [32.58  ] */
    getch();
}
```

ในตัวอย่างนี้มีการใช้ฟังก์ชันเพิ่มเติมเพื่อช่วยให้เห็นผลลัพธ์ของการแสดงผลที่จอภาพ) ในกรณีที่ผู้เขียนโปรแกรมใช้คอมไพเลอร์ Turbo C คือ

- clrscr() ใช้เพื่อให้ลบข้อความต่าง ๆ ที่ค้างบนจอภาพก่อนที่จะแสดงผลโปรแกรม
- getch() ใช้เพื่อรอรับการกดแป้นพิมพ์ใด ๆ หลังจากแสดงผลแล้ว เพื่อให้เกิดการค้างรอให้ผู้ใช้กดแป้นพิมพ์ก่อนจะไปทำงานอื่นต่อไป

ทั้ง 2 ฟังก์ชันอยู่ในฟังก์ชันมาตรฐานของคอมไพเลอร์ Turbo C ต้องเรียกใช้อินคลูซไฟล์ชื่อ conio.h ตัวอย่างที่ 1.6 เป็นตัวอย่างของการรับข้อมูลเข้าเพื่อแสดงผลทางจอภาพ

ตัวอย่างที่ 1.6 แสดงตัวอย่างการรับข้อมูลเพื่อนำมาแสดงผล

```
#include <stdio.h>

void main() {
    char name[100];

    printf("What is your name ?\n");
    scanf("%s", name);
    printf("Very glad to know you, ");
    printf("%s.", name);
}
```

ผลลัพธ์ของการทำงาน

```
What is your name ?
Willy
Very glad to know you, Willy.
```

หมายเหตุ ตัวเอียงคือข้อความที่ผู้ใช้ป้อนเข้าสู่ระบบ

ข้อควรระวัง

- การป้อนข้อมูลที่เป็นสตริงไม่จำเป็นต้องใส่เครื่องหมาย & นำหน้าตัวแปรสตริง
 - หากชื่อที่ป้อนเข้าสู่โปรแกรมมีช่องว่าง เช่นผู้ใช้ป้อนชื่อและนามสกุล โดยมีช่องว่างระหว่างชื่อและนามสกุล โปรแกรมจะรับได้เฉพาะข้อมูลชื่อเท่านั้น โปรแกรมไม่สามารถรับข้อมูล ที่มีช่องว่างเข้ามาเก็บในตัวแปรได้
-

5. ให้ประกาศตัวแปรเพื่อใช้เก็บข้อมูลต่าง ๆ ต่อไปนี้
- (ก) เลขจำนวนเต็ม 5 จำนวน เก็บค่าตัวเลขไม่เกิน 100,000
 - (ข) ความสูง มีหน่วยเป็นเซนติเมตร
 - (ค) นามสกุล ความยาวไม่เกิน 30 ตัวอักษร
 - (ง) เลขที่บัตรประจำตัวประชาชนเป็นตัวเลข 13 หลัก
 - (จ) จำนวนพนักงานในบริษัทแห่งหนึ่ง มีพนักงานไม่เกิน 500 คน
 - (ฉ) คะแนนสอบของนักศึกษาในกระบวนวิชาหนึ่ง มีค่าไม่เกิน 100 คะแนน
 - (ช) เงินเดือนที่พนักงานบริษัทแห่งหนึ่งได้รับ
6. ให้เขียนโปรแกรมเพื่อรับข้อมูลต่าง ๆ ในข้อ 5 และแสดงผลข้อมูลที่ได้รับนั้น
7. ให้เขียนโปรแกรม 2 โปรแกรมเพื่อแสดงข้อความต่อไปนี้ออกทางจอภาพ
- ```
I love
C programming
very much.
```
- โดยที่
- (ก) เขียนโปรแกรมที่ 1 โดยในฟังก์ชัน main() ประกอบด้วยฟังก์ชัน printf() 3 คำสั่ง
  - (ข) เขียนโปรแกรมที่ 2 โดยใช้ฟังก์ชัน main() ประกอบด้วยฟังก์ชัน printf() 1 คำสั่ง
8. ให้เขียนโปรแกรมเพื่อรับชื่อและความสูงของคน ๆ หนึ่ง และแสดงชื่อและความสูงนั้นทางจอภาพ โดยแสดงในรูปแบบดังตัวอย่างต่อไปนี้ (ตัวเอนคือสิ่งที่ผู้ใช้ป้อนสู่ระบบ)
- ```
Enter name : Somchai
Enter height (cm.) : 178
Output :
Name          Height (cm.)
Somchai       178
```
9. ให้เขียนโปรแกรมรับข้อมูลเลขจำนวนจริงแบบ float 3 จำนวน และแสดงผลลัพธ์เลขทั้ง 3 จำนวนดังตัวอย่าง (ตัวเอนคือสิ่งที่ผู้ใช้ป้อนสู่ระบบ)
- ```
Enter number 1 : 5243.2
Enter number 2 : 13
Enter number 3 : 12.3548
Output :
5243.20
13.00
12.35
```



## (Operators and Expressions)

ในการเขียนโปรแกรมองค์ประกอบที่สำคัญสิ่งหนึ่งคือ ตัวดำเนินการ ซึ่งมักจะนำใช้ร่วมกับนิพจน์ในลักษณะต่าง ๆ ตัวดำเนินการที่สำคัญ ได้แก่ ตัวดำเนินการกำหนดค่า (Assignment Operator) ตัวดำเนินการคณิตศาสตร์ (Arithmetic Operators) ตัวดำเนินการกำหนดค่าแบบผสม (Compound Assignment Operators) ตัวดำเนินการเพิ่มค่าและลดค่า (Increment and Decrement Operators) ตัวดำเนินการเปลี่ยนชนิดข้อมูล (Type Cast Operator) ตัวดำเนินการความสัมพันธ์ (Relational Operators) ตัวดำเนินการความเท่ากัน (Equality Operators) ตัวดำเนินการตรรกะ (Logical Operators) ตัวดำเนินการเงื่อนไข (Conditional Operator) ตัวดำเนินการคอมมา (Comma Operator)

ตัวดำเนินการเหล่านี้เมื่อนำไปใช้กับนิพจน์จะทำให้เกิดนิพจน์ในรูปแบบต่าง ๆ เช่น นิพจน์กำหนดค่า นิพจน์คณิตศาสตร์ นิพจน์ตรรกศาสตร์ เป็นต้น

### 1. ตัวดำเนินการกำหนดค่า

ตัวดำเนินการกำหนดค่าเป็นตัวดำเนินการพื้นฐานที่ใช้ในการกำหนดค่าต่าง ๆ ให้กับตัวแปร โดยใช้เครื่องหมาย = มีรูปแบบของนิพจน์กำหนดค่าคือ

ตัวแปร = นิพจน์ ;

โดยที่นิพจน์อาจจะเป็นค่าคงที่ ตัวแปร หรือนิพจน์ที่ประกอบขึ้นจากตัวดำเนินการต่าง ๆ ก็ได้ตัวอย่างเช่น

```
age = 10;
speed = distance / time;
total_min = 60 * hours + minutes;
ch = 'A';
```

ความหมายของตัวดำเนินการกำหนดค่าคือ การกำหนดค่าที่อยู่ทางขวามือของคำสั่งให้กับตัวแปรที่อยู่ทางซ้ายมือ

นอกจากนี้ยังมีรูปแบบเพิ่มเติมหากต้องการกำหนดค่าให้กับตัวแปรหลายตัวด้วยค่าเดียวกัน มีรูปแบบดังนี้คือ

ตัวแปร1 = ตัวแปร2 = ... = นิพจน์ ;

โดยตัวแปรทั้งหมดจะต้องเป็นตัวแปรประเภทเดียวกัน เช่น

```
int x, y, z;
x = y = z = 45;
```

แปลความหมายได้ว่า

```
z = 45;
y = z;
x = y;
```

นอกจากนี้เมื่อมีการใช้ตัวดำเนินการกำหนดค่าจะมีการแปลงค่าชนิดของข้อมูลโดยอัตโนมัติอีกด้วย (Implicit Casting) ซึ่งจะเกิดกับข้อมูลตัวเลข เช่น กำหนดค่าที่มีชนิดเป็น int ให้กับตัวแปรที่มีชนิดเป็น float จะเกิดกระบวนการแปลงค่าจาก int ไปเป็น float เช่น

```
float x;
x = 14;
```

14 เป็นข้อมูลประเภท int แต่ x เป็นตัวแปรประเภท float เพราะฉะนั้นค่า 14 จะถูกแปลงเป็น 14.0f ก่อนที่จะนำไปเก็บยังตัวแปร x แต่หากมีการกำหนดค่าที่มีชนิดเป็นจำนวนจริงให้กับตัวแปรที่เป็นจำนวนเต็มจะเกิดการบิตเศษทิ้ง ตัวอย่างเช่น

```
int a;
a = 3.5;
```

จะได้ว่ามีการแปลงค่า 3.5 ไปเป็น 3 ก่อนที่จะนำค่านั้นไปกำหนดให้กับ a จะได้ว่า a เก็บค่า 3 พิจารณาตัวอย่างต่อไปนี้

```
int a;
float x;
x = a = 12.74;
```

จะได้ว่ามีการกำหนดค่า a = 12.34 ก่อน ซึ่งมีการแปลงค่า 12.74 โดยบิตเศษทิ้ง ได้ว่า a เก็บค่า 12 จากนั้นนำค่าใน a ไปกำหนดค่าให้กับตัวแปร x ซึ่งเป็น float จะมีการแปลงค่า 12 เป็น 12.0f จะได้ว่า x เก็บค่า 12.0 แสดงตัวอย่างเรื่องการบิตเศษด้วยตัวอย่างที่ 2.1

## ตัวอย่างที่ 2.1 แสดงการกำหนดค่าจำนวนจริงให้กับตัวแปรจำนวนเต็ม

```
#include <stdio.h>

void main() {
 int x;
 x = 14.8328;
 printf("x value is %d", x);
}
```

### ผลการทำงานของโปรแกรม

x value is 14

## 2. ตัวดำเนินการคณิตศาสตร์

ตัวดำเนินการคณิตศาสตร์ในภาษาซีประกอบด้วยการบวก ลบ คูณ หาร หารเอาเศษ การเขียนนิพจน์คณิตศาสตร์ทางคอมพิวเตอร์จะแตกต่างกับนิพจน์คณิตศาสตร์ที่เคยเรียนทั่วไป ตรงที่มีลำดับความสำคัญ (Precedence) ของตัวดำเนินการเข้ามาเกี่ยวข้อง แสดงตัวดำเนินการคณิตศาสตร์และการทำงานดังตาราง 2.1 และแสดงลำดับความสำคัญของตัวดำเนินการดังตาราง 2.2

ตาราง 2.1 แสดงตัวดำเนินการคณิตศาสตร์และการทำงาน

| ตัวดำเนินการ | คำอธิบาย       | ตัวอย่าง | การทำงาน  | ผลลัพธ์ |
|--------------|----------------|----------|-----------|---------|
| +            | Unary plus     | +10      | ขวาไปซ้าย | +10     |
| -            | Unary minus    | -7       | ขวาไปซ้าย | -7      |
| *            | Multiplication | 10*3     | ซ้ายไปขวา | 30      |
| /            | Division       | 10/3     | ซ้ายไปขวา | 3       |
| %            | Modulus        | 10%3     | ซ้ายไปขวา | 1       |
| +            | Addition       | 10+3     | ซ้ายไปขวา | 13      |
| -            | Subtraction    | 10-3     | ซ้ายไปขวา | 7       |

ตาราง 2.2 แสดงลำดับความสำคัญของตัวดำเนินการ

| ตัวดำเนินการ | คำอธิบาย                          |
|--------------|-----------------------------------|
| +, -         | Unary Plus, Unary Minus           |
| *, /, %      | Multiplication, Division, Modulus |
| +, -         | Addition, Subtraction             |

จากตาราง 2.2 ตัวดำเนินการที่อยู่ด้านบนจะมีความสำคัญว่าด้านล่าง นั่นคือหากในนิพจน์ประกอบด้วยตัวดำเนินการด้านบนจะมีการแปลค่าตัวดำเนินการนั้นการ หากพบตัวดำเนินการที่อยู่ในลำดับเดียวกันก็จะทำการแปลความหมายจากซ้ายไปขวา ตัวอย่างเช่น

$$3 + 4 / 2 \quad \text{จะได้ผลลัพธ์เท่ากับ } 3 + (4 / 2) = 5$$

$$3 * 2 + 4 \% 2 \quad \text{จะได้ผลลัพธ์เท่ากับ } (3 * 2) + (4 \% 2) = 6 + 0 = 6$$

แต่หากต้องการให้ทำตัวดำเนินการในลำดับต่ำกว่าก่อน ให้ใช้เครื่องหมาย ( ) ครอบคำสั่งที่ต้องการ เช่น

$$3 * (2 + 4) \% 2 \quad \text{จะได้ผลลัพธ์เท่ากับ } 3 * 6 \% 2 = 18 \% 2 = 0$$

เพราะฉะนั้นหากมีสมการเช่น

$$X = \frac{A + B + C}{10}$$

หากต้องการเขียนเป็นนิพจน์คณิตศาสตร์ จะต้องเขียนว่า

$$X = (A + B + C) / 10$$

**สิ่งสำคัญ**อีกสิ่งหนึ่งในการใช้ตัวดำเนินการคณิตศาสตร์ คือ การคำนวณภายในนิพจน์ที่เกิดจากข้อมูลชนิดจำนวนเต็มกับข้อมูลจำนวนจริง และการใช้ Modulus ตัวอย่างเช่น

```
float f;
f = 17.0 / 4; /* f = 4.25 */
f = 17 / 4.0; /* f = 4.25 */
f = 17 / 4; /* f = 4.0 */
```

```
int a;
a = -17 / 3; /* a = -5 */
a = 17 / -3; /* a = -5 */
a = -17 / -3; /* a = 5 */
```

```
a = 17 % 3; /* a = 2 */
a = 15 % 3; /* a = 0 */
```

```
a = -17 % 3; /* a = -2 */
a = 17 % -3; /* a = 2 */
a = -17 % -3; /* a = -2 */
```

และแสดงตัวอย่างการทำงานของตัวดำเนินการคณิตศาสตร์ดังตาราง 2.3 สมมติให้ a และ b มีค่าเป็นจำนวนเต็ม และแสดงตัวอย่างการหาผลรวมของเลขจำนวนเต็ม 2 จำนวนที่รับจากผู้ใช้ ดังตัวอย่างที่ 2.2

ตาราง 2.3 แสดงตัวอย่างการประมวลผลตัวดำเนินการคณิตศาสตร์

| a   | b  | a/b | a%b | a + b * 3 |
|-----|----|-----|-----|-----------|
| 17  | 3  | 5   | 2   | 26        |
| -17 | 3  | -5  | -2  | -8        |
| 17  | -3 | -5  | 2   | 8         |
| -17 | -3 | 5   | -2  | -26       |

### ตัวอย่างที่ 2.2 โปรแกรมหาผลรวมของเลขจำนวนเต็ม 2 จำนวนที่รับข้อมูลจากผู้ใช้

```
#include <stdio.h>
void main() {
 int x, y, z;
 printf("Enter X value : ");
 scanf("%d", &x);
 printf("Enter Y value : ");
 scanf("%d", &y);
 z = x + y;
 printf("Summary of X and Y is %d", z);
}
```

#### ผลการทำงานของโปรแกรม

```
Enter X value : 15
Enter Y value : 20
Summary of X and Y is 35
```

### 3. ตัวดำเนินการกำหนดค่าแบบผสม

ตัวดำเนินการกำหนดค่าแบบผสมเป็นตัวดำเนินการที่ผสมระหว่างตัวดำเนินการกำหนดค่าและตัวดำเนินการคณิตศาสตร์ แสดงดังตาราง 2.4

ตาราง 2.4 แสดงตัวดำเนินการกำหนดค่าแบบผสม

| ตัวดำเนินการ | ตัวอย่างคำสั่ง | คำสั่งเต็ม |
|--------------|----------------|------------|
| *=           | a *= 2;        | a = a * 2; |
| /=           | a /= 2;        | a = a / 2; |
| %=           | a %= 2;        | a = a % 2; |
| +=           | a += 2;        | a = a + 2; |
| -=           | a -= 2;        | a = a - 2; |

การแปลคำสั่งของตัวดำเนินการกำหนดค่าแบบผสม จะทำคำสั่งที่อยู่ทางขวามือก่อนเสมอ เช่น

$$a *= 3 + 2;$$

จะเท่ากับคำสั่ง

$$a = a * (3 + 2);$$

#### 4. ตัวดำเนินการเพิ่มค่าและลดค่า

ตัวดำเนินการเพิ่มค่าและลดค่า เป็นตัวดำเนินการเพื่อใช้เพิ่มค่าตัวแปรขึ้น 1 หรือลดค่าตัวแปรลง 1 โดยใช้เครื่องหมาย ++ แทนการเพิ่มค่าขึ้น 1 และ -- แทนการลดค่าลง 1 และสามารถในตัวดำเนินการเพิ่มค่าหรือลดค่ากับตัวแปรได้ 2 ตำแหน่ง คือ วางตัวดำเนินการเพิ่มค่าหรือลดค่าไว้หน้าตัวแปร และวางไว้หลังตัวแปร ดังตัวอย่าง

$$a++;$$

$$++a;$$

ทั้ง 2 คำสั่งจะมีค่าเท่ากับ  $a = a + 1$ ; ส่วนคำสั่ง

$$a--;$$

$$--a;$$

$$\text{จะมีค่าเท่ากับ } a = a - 1;$$

#### ความแตกต่างของตำแหน่งการวางตัวดำเนินการเพิ่มค่าหรือลดค่า

ตำแหน่งของการวางตัวดำเนินการเพิ่มค่าหรือลดค่าจะส่งผลถึงการทำงานของคำสั่งนั้น ตัวอย่างเช่น

$$b = 5;$$

$$a = 10 + b++ * 2;$$

ผลลัพธ์ที่ได้ จะมีผลเท่ากับคำสั่ง

$$a = 10 + b * 2;$$

$$b = b + 1;$$

$$\text{จะได้ว่า } a \text{ มีค่าเท่ากับ } 10 + 5 * 2 = 20 \text{ และ } b \text{ มีค่าเท่ากับ } 6$$

แต่หากนำตัวดำเนินการเพิ่มค่ามาไว้ด้านหน้าตัวแปร ดังตัวอย่าง โดยกำหนดให้ b มีค่าเท่ากับ 5 เช่นเดียวกัน

$$b = 5;$$

$$a = 10 + ++b * 2;$$

ผลลัพธ์ที่ได้ จะมีผลเท่ากับคำสั่ง

$b = b + 1;$

$a = 10 + b * 2;$

จะได้  $b$  มีค่าเท่ากับ 6 และ  $a$  มีค่าเท่ากับ  $10 + 6 * 2 = 22$

### ตัวอย่างที่ 2.3 แสดงการใช้ตัวดำเนินการเพิ่มค่า

```
#include <stdio.h>
void main() {
 int y, count;
 count = 1;
 y = count++;
 printf("y = %d, count = %d", y, count);
 count = 1;
 y = ++count;
 printf("\ny = %d, count = %d", y, count);
}
```

#### ผลการทำงานของโปรแกรม

y = 1, count = 2

y = 2, count = 2

## 5. ตัวดำเนินการเปลี่ยนชนิดของข้อมูล

การเปลี่ยนชนิดข้อมูลในภาษาซีมี 2 ลักษณะ คือ การเปลี่ยนชนิดข้อมูลโดยไม่ต้องใช้คำสั่ง (Implicit Casting) ซึ่งกล่าวถึงในหัวข้อที่ 2.1 และการเปลี่ยนชนิดข้อมูลโดยใช้คำสั่ง (Explicit Casting) เช่น หากต้องแปลงข้อมูลชนิด float ไปเป็นอีกข้อมูลชนิด int จะต้องใช้คำสั่ง

int a;

a = (int)12.423;

จะได้ว่า  $a$  มีค่าเท่ากับ 12 กระบวนการทำงานจะมีการเปลี่ยนชนิดข้อมูลที่อยู่ใกล้กับตัวดำเนินการเปลี่ยนชนิดข้อมูลให้เป็นชนิดข้อมูลที่ระบุในวงเล็บ แล้วจึงมีการกำหนดค่าใหม่นั้นให้กับ  $a$  ทั้งนี้สามารถกำหนดชนิดข้อมูลที่จะเปลี่ยนค่าเป็นชนิดข้อมูลใด ๆ ก็ได้ แสดงตัวอย่างเพิ่มเติมดังตัวอย่างที่ 2.4

## ตัวอย่างที่ 2.4 แสดงการใช้ตัวดำเนินการเปลี่ยนชนิดข้อมูล

```
#include <stdio.h>
void main() {
 int x;
 x = 2.5 * 2;
 printf("x value is %d", x);
 x = (int)2.5 * 2;
 printf("\nx value is %d", x);
 x = (int)(2.5 * 2);
 printf("\nx value is %d", x);
}
```

### ผลการทำงานของโปรแกรม

```
x value is 5
x value is 4
x value is 5
```

## 6. ตัวดำเนินการคอมมา

ตัวดำเนินการคอมมา เป็นตัวดำเนินการเพื่อบอกถึงลำดับการทำงาน เพื่อช่วยกำหนดการทำงานของนิพจน์หนึ่ง ๆ มักจะใช้ร่วมกับคำสั่งควบคุม รูปแบบการใช้ตัวดำเนินการคอมมา คือ

นิพจน์1 , นิพจน์2

การทำงานจะทำงานที่นิพจน์ทางซ้ายมือก่อนเสมอ แล้วจึงทำงานที่นิพจน์ขวามือ แต่ถ้าใช้คำสั่ง

$$s = (t = 2, t + 3);$$

ลำดับการทำงานคือ  $t = 2$  หลังจากนั้นจะนำค่าใน  $t$  คือ 2 ไปบวกกับ 3 จะได้ค่าคือ 5 และนำค่า 5 ไปกำหนดให้กับ  $s$  จะได้ว่า  $t$  มีค่า 2 และ  $s$  มีค่า 5

.....  
แต่หากไม่มีเครื่องหมายวงเล็บดังตัวอย่าง

$$s = t = 2, t + 3$$

จะได้ว่า  $s$  เท่ากับ 2 และ  $t$  เท่ากับ 2 หลังจากนั้นนำค่า  $t$  ไปบวก 3 ได้ผลลัพธ์ของนิพจน์เป็น 5 มักจะใช้กับนิพจน์ตรรกศาสตร์ในคำสั่งควบคุมแบบต่าง ๆ ซึ่งจะได้กล่าวถึงในบทที่ 3



## 7. ตัวดำเนินการความสัมพันธ์

ตัวดำเนินการความสัมพันธ์ได้แก่ตัวดำเนินการที่ใช้เปรียบเทียบนิพจน์ 2 นิพจน์ มักใช้กับคำสั่งควบคุม ซึ่งจะกล่าวถึงในบทที่ 3 ดังตาราง 2.5

ตาราง 2.5 แสดงตัวดำเนินการความสัมพันธ์

| ตัวดำเนินการ | ความหมาย            |
|--------------|---------------------|
| >            | มากกว่า             |
| <            | น้อยกว่า            |
| >=           | มากกว่าหรือเท่ากับ  |
| <=           | น้อยกว่าหรือเท่ากับ |

ผลที่ได้จากการใช้ตัวดำเนินการความสัมพันธ์ คือ จริง (True) หรือเท็จ (False) ซึ่งในภาษาซีแทนด้วยเลขจำนวนเต็ม กรณีเท็จจะแทนด้วยค่า 0 และกรณีจริงจะแทนด้วยค่าที่ไม่ใช่ 0 ทดสอบค่าการเปรียบเทียบดังตัวอย่างที่ 2.5 และแสดงตัวอย่างการประมวลผลนิพจน์ความสัมพันธ์ดังตาราง 2.6

### ตัวอย่างที่ 2.5 แสดงค่าของการเปรียบเทียบด้วยตัวดำเนินการความสัมพันธ์

```
#include <stdio.h>
void main() {
 int x, y;
 printf("Enter X : ");
 scanf("%d", &x);
 printf("Enter Y : ");
 scanf("%d", &y);
 printf("X > Y is %d", x>y);
}
```

#### ผลการทำงานของโปรแกรม

```
Enter X : 32
Enter Y : 24
X > Y is 1
```

ตาราง 2.6 แสดงตัวอย่างการประมวลผลนิพจน์ความสัมพันธ์ กำหนดให้  $a = 5$  และ  $b = 3$ 

| นิพจน์            | แปลงนิพจน์          | ค่าตรรกะ | ค่าที่ได้ |
|-------------------|---------------------|----------|-----------|
| $5+2*4 < (5+2)*4$ | $5+(2*4) < (5+2)*4$ | True     | 1         |
| $a + b <= b+a$    | $(5+3) <= (3+5)$    | True     | 1         |
| $a/b < b/a$       | $(5/3) < (3/5)$     | False    | 0         |

## 8. ตัวดำเนินการความเท่ากัน

ตัวดำเนินการความเท่ากันเป็นตัวดำเนินการเพื่อใช้เปรียบเทียบความเท่ากันของนิพจน์ 2 นิพจน์ มักใช้กับคำสั่งควบคุมซึ่งจะกล่าวถึงในบทที่ 3 มีตัวดำเนินการดังตาราง 2.7

ตาราง 2.7 แสดงตัวดำเนินการความเท่ากัน

| ตัวดำเนินการ    | ความหมาย   |
|-----------------|------------|
| <code>==</code> | เท่ากัน    |
| <code>!=</code> | ไม่เท่ากัน |

ผลลัพธ์ของการเปรียบเทียบมีค่าคือจริง หรือเท็จ การใช้งานจะต้องระวังเพราะมีความสับสนระหว่างการใช้ตัวดำเนินการความเท่ากัน `==` กับตัวดำเนินการกำหนดค่า `=` ซึ่งมีการทำงานที่ต่างกัน และตัวดำเนินการไม่เท่ากันใช้เครื่องหมาย `!=` ไม่ใช่เครื่องหมาย `<>` เหมือนในภาษาอื่น ๆ เช่น

`a == 2` เป็นการเปรียบเทียบว่าตัวแปร `a` มีค่าเท่ากับ 2 หรือไม่

`a = 2` เป็นการกำหนดค่า 2 ให้กับตัวแปร `a`

ในการเปรียบเทียบค่าจะต้องระวังเรื่องของเลขทศนิยมซึ่งเกิดจากการคำนวณของเครื่องคอมพิวเตอร์ ซึ่งคอมพิวเตอร์แต่ละเครื่องอาจจะให้ผลการคำนวณที่ต่างกัันดังตัวอย่าง

```
float a=1.0f;
```

คำสั่ง `a == a / 3.0 * 3.0` อาจจะให้ค่าถ้าคาดไม่ถึง เนื่องจากในทางคณิตศาสตร์  $1/3 * 3$  จะได้ค่าเท่ากับ 1 แต่ในทางคอมพิวเตอร์ เครื่องคอมพิวเตอร์บางเครื่องเมื่อ  $1/3$  จะได้ค่า 0.33333... เมื่อนำกลับมาคูณกับ 3 จะได้ค่า 0.99999... ซึ่งผลลัพธ์จะไม่เท่ากับ 1.0 ตามที่ผู้เขียนโปรแกรมตั้งใจ แสดงตัวอย่างการใช้ตัวดำเนินการความเท่ากันดังตัวอย่างที่ 2.6

## ตัวอย่างที่ 2.6 แสดงค่าของการเปรียบเทียบด้วยตัวดำเนินการความเท่ากัน

```
#include <stdio.h>

void main() {
 int x, y, result;
 printf("Enter X : ");
 scanf("%d", &x);
 printf("Enter Y : ");
 scanf("%d", &y);
 result = (x==y);
 printf("X == Y is %d", result);
}
```

### ผลการทำงานของโปรแกรม

```
Enter X : 25
Enter Y : 4
X == Y is 0
```

## 9. ตัวดำเนินการตรรกะ

ตัวดำเนินการตรรกะเป็นตัวดำเนินการที่ใช้คู่กับตัวดำเนินการความสัมพันธ์และตัวดำเนินการความเท่ากันซึ่งมักจะใช้กับคำสั่งควบคุมซึ่งจะกล่าวถึงในบทที่ 3 มีตัวดำเนินการดังตาราง 2.8 และมีการทำงานเหมือนกับการเปรียบเทียบเชิงตรรกะทั่วไป แสดงด้วยตารางค่าความจริงในตาราง 2.9

ตาราง 2.8 แสดงตัวดำเนินการตรรกะ

| ตัวดำเนินการ | ความหมาย    |
|--------------|-------------|
| &&           | Logical AND |
|              | Logical OR  |
| !            | Logical NOT |

ตาราง 2.9 แสดงตารางค่าความจริงของตัวดำเนินการตรรกะ

| P     | Q     | P&&Q  | P  Q  | P     | !P    |
|-------|-------|-------|-------|-------|-------|
| true  | true  | true  | true  | true  | false |
| true  | false | false | true  | false | true  |
| false | true  | false | true  |       |       |
| false | false | false | false |       |       |

ตัวอย่างของการเปรียบเทียบได้แก่

```
!10 ผลลัพธ์คือ 0
!0 ผลลัพธ์คือ 1
!'A' ผลลัพธ์คือ 0
```

สมมติให้ `int a=1, b=3, c=4;`

แสดงตัวอย่างการประมวลผลดังตาราง 2.10

ตาราง 2.10 แสดงตัวอย่างการใช้ตัวดำเนินการตรรกะ

| นิพจน์                                                | แปลงนิพจน์                                          | ค่าตรรกะ | ค่าที่ได้ |
|-------------------------------------------------------|-----------------------------------------------------|----------|-----------|
| <code>a + b &gt; 0 &amp;&amp; b / c &gt; 1</code>     | <code>((a+b)&gt;0) &amp;&amp; ((b/c) &gt; 1)</code> | False    | 0         |
| <code>a / c == b / c &amp;&amp; (a+b) / c == 1</code> | <code>((a/c)==(b/c))&amp;&amp;(((a+b)/c)==1)</code> | True     | 1         |
| <code>!a    !b    !c</code>                           | <code>(!a)    (!b)    (!c)</code>                   | False    | 0         |

ในการใช้ตัวดำเนินการตรรกะสิ่งที่ต้องระวังคือ Short Circuit เนื่องจากเราทราบว่า ในกรณีของ `&&` (Logical And) หากนิพจน์แรกเป็นเท็จ ไม่ว่านิพจน์ที่ 2 เป็นอะไร จะทำให้นิพจน์นั้นเป็นเท็จเสมอ และในกรณีของ `||` (Logical Or) หากนิพจน์แรกเป็นจริง ไม่ว่านิพจน์ที่ 2 เป็นอะไร จะทำให้นิพจน์นั้นเป็นจริงเสมอ คอมไพเลอร์ใช้หลักการคิดนี้เช่นเดียวกัน ทำให้ไม่มีการประมวลผลนิพจน์ที่ 2 ในกรณีที่นิพจน์แรกทำให้เกิด Short Circuit แสดงดังตัวอย่าง

```
int a=10;
printf("[%d]", a < 10 && a++ > 10);
printf("\na is %d", a);
```

จากตัวอย่างผู้เขียนต้องการเพิ่มค่า `a` ขึ้น 1 หลังจากทำงานฟังก์ชัน `printf( )` คำสั่งแรก แต่เนื่องจากนิพจน์ `a < 10` เป็นเท็จ ทำให้นิพจน์หลังไม่ถูกประมวลผล จึงไม่เกิดการเพิ่มค่า `a` จะได้ `a` มีค่าเท่ากับ 10 เหมือนเดิม

พิจารณาตัวอย่าง Short Circuit กรณีของการใช้ตัวดำเนินการ `||`

```
int a = 10;
printf("[%d]", a < 20 || a++ > 10);
printf("\na is %d", a);
```

จากตัวอย่างนิพจน์ `a < 20` เป็นจริง ทำให้เกิด Short Circuit คือไม่มีการประมวลผลในนิพจน์ที่ 2 ของตัวดำเนินการ `||` ค่า `a` จึงไม่ถูกเพิ่มค่า จะได้คำตอบว่า `a` มีค่าเท่ากับ 10 เหมือนเดิม

นอกจากนี้การเขียนนิพจน์ตรรกศาสตร์ที่คุ้นเคยในชีวิตประจำวันในบางลักษณะไม่สามารถทำได้ในทางคอมพิวเตอร์ ตัวอย่างเช่น หากต้องการทราบว่า  $x$  มีค่าอยู่ในช่วงตั้งแต่ 10 ถึง 20 จริงหรือไม่ ปกติแล้วจะเขียนว่า  $10 \leq x \leq 20$  ซึ่งรูปแบบนี้ไม่สามารถใช้ได้ทางคอมพิวเตอร์ ที่ถูกต้องจะต้องเขียนว่า

$$x \geq 10 \ \&\& \ x \leq 20$$

## 10. ตัวดำเนินการเงื่อนไข

ตัวดำเนินการเงื่อนไขเป็นตัวดำเนินการที่ใช้ตรวจสอบเงื่อนไขของนิพจน์ มีรูปแบบคือ

$$\text{นิพจน์1} \ ? \ \text{นิพจน์2} \ : \ \text{นิพจน์3}$$

การประมวลผลจะตรวจสอบนิพจน์1 ว่าเป็นจริงหรือเท็จ หากเป็นจริงจะทำงานในนิพจน์2 แต่หากเป็นเท็จจะทำงานในนิพจน์3 แสดงดังตัวอย่าง

$$\text{min} = (a < b) \ ? \ a \ : \ b;$$

การดำเนินการจะเปรียบเทียบค่า  $a$  และ  $b$  หาก  $a$  มีค่าน้อยกว่า  $b$  จริง จะได้ว่า  $\text{min} = a$  แต่หากนิพจน์  $a < b$  เป็นเท็จจะได้ว่า  $\text{min} = b$ ; แสดงตัวอย่างเพิ่มเติมในตัวอย่างที่ 2.7 และ 2.8

**ตัวอย่างที่ 2.7** โปรแกรมรับข้อมูลเลขจำนวนเต็มจากผู้ใช้ 2 จำนวน คือ  $x$  และ  $y$  หาก  $x$  มีค่ามากกว่า  $y$  ให้ขึ้นข้อความว่า “X more than Y” แต่ถ้าไม่ใช่ให้ขึ้นข้อความว่า “X not more than Y”

---

```
#include <stdio.h>
void main() {
 int x, y;
 printf("Enter X : ");
 scanf("%d", &x);
 printf("Enter Y : ");
 scanf("%d", &y);
 (x > y) ? printf("X more than Y") : printf("X not more than Y");
}
```

---

### ผลการทำงานของโปรแกรม

Enter X : 12

Enter Y : 8

X more than Y

---

**ตัวอย่างที่ 2.8** โปรแกรมรับข้อมูลเลขจำนวนเต็มจากผู้ใช้ 2 จำนวน คือ x และ y ให้หาว่าค่าที่มีค่าน้อยที่สุดคือค่าใด

---

```
#include <stdio.h>
void main() {
 int x, y, min;
 printf("Enter X : ");
 scanf("%d", &x);
 printf("Enter Y : ");
 scanf("%d", &y);
 min = (x < y) ? x : y;
 printf("Minimum value is %d", min);
}
```

---

---

#### ผลการทำงานของโปรแกรม

Enter X : *12*

Enter Y : *8*

Minimum value is 8

---

---



d = a + b \* 3 / 2 - b / 4 + a % 2 + 10; /\* d = \_\_\_\_\_ \*/

d = 5

d += a + b % 3; /\* d = \_\_\_\_\_ \*/

d %= b - 3; /\* d = \_\_\_\_\_ \*/

d = a+++ + b+++ + 2; /\* d = \_\_\_\_\_ \*/

/\* a = \_\_\_\_\_ \*/

/\* b = \_\_\_\_\_ \*/

d = ++a + a++; /\* d = \_\_\_\_\_ \*/

/\* a = \_\_\_\_\_ \*/

d = a + b+++ 2 - b+++ / 4 + ++a % 2; /\* d = \_\_\_\_\_ \*/

/\* a = \_\_\_\_\_ \*/

/\* b = \_\_\_\_\_ \*/

(ข) int a=5, b;

float x=12.5, y;

b = 15.7; /\* b = \_\_\_\_\_ \*/

y = 10; /\* y = \_\_\_\_\_ \*/

b = a / 2; /\* b = \_\_\_\_\_ \*/

y = x / 2; /\* y = \_\_\_\_\_ \*/

b = a % 3 \* 2.5; /\* b = \_\_\_\_\_ \*/

y = x + 6 / 3; /\* y = \_\_\_\_\_ \*/

(ค) int a=5, b=10, c=3;

a > b || a < c /\* คำตอบ = \_\_\_\_\_ \*/

(a + b / 3) > 10 /\* คำตอบ = \_\_\_\_\_ \*/

a\*c > b && b\*a/c < b/a\*c /\* คำตอบ = \_\_\_\_\_ \*/

a >= 10 || b >= 10 && c\*a >= 10 /\* คำตอบ = \_\_\_\_\_ \*/

!(a < b) && (a > c) || !(a+c > b) /\* คำตอบ = \_\_\_\_\_ \*/

5. เขียนโปรแกรมคำนวณพื้นที่ของวงกลม โดยรับรัศมีของวงกลมจากผู้ใช้ สูตรการหาพื้นที่ของวงกลมได้แก่

$$CircleArea = \frac{1}{2} PI * r^2$$

กำหนดให้ค่า PI คือ 3.14159265

6. เขียนโปรแกรมรับข้อมูลจำนวนเต็ม 5 จำนวนจากผู้ใช้ และหาค่าเฉลี่ยของข้อมูลที่ได้รับเข้ามามีค่าเท่าใด
7. เขียนโปรแกรมให้รับค่าจำนวนจริงจากผู้ใช้ 1 จำนวน และให้หาค่าเลขดังกล่าวอยู่ในช่วงของเลขจำนวนเต็มใด เช่น หากผู้ใช้ป้อนเลข 12.5 ให้ตอบว่า "12.5 is between 12 and 13" (ใช้ตัวดำเนินการเปลี่ยนชนิดข้อมูลในการเขียนโปรแกรม)
8. เขียนโปรแกรมเพื่อรับข้อมูลเลขจำนวนจริงจากผู้ใช้ 3 จำนวน ให้หาค่าที่มากที่สุดที่ป้อนเข้ามาคือค่าใด โดยใช้ตัวดำเนินการเงื่อนไข



## (Control Statements)

ในการเขียนโปรแกรมแบบโครงสร้าง จะมีรูปแบบการแก้ปัญหาหรือรูปแบบการเขียนโปรแกรมอยู่ 3 ลักษณะ คือ การเขียนแบบลำดับ (Sequential) การเขียนแบบเงื่อนไข (Selection) และการเขียนแบบวนซ้ำ (Repetition) การเขียนทีละคำสั่งจากบนลงจัดเป็นการเขียนแบบลำดับ ส่วนการเขียนแบบเงื่อนไขและการเขียนแบบวนซ้ำนั้นจะต้องใช้คำสั่งควบคุมมาช่วยให้เกิดการเขียนในลักษณะดังกล่าว โดยที่ใช้ภาษาซีมีคำสั่ง if และ switch เพื่อช่วยในการเขียนแบบเงื่อนไข ส่วนในการเขียนแบบวนซ้ำจะมี 3 คำสั่งคือ for while และ do-while

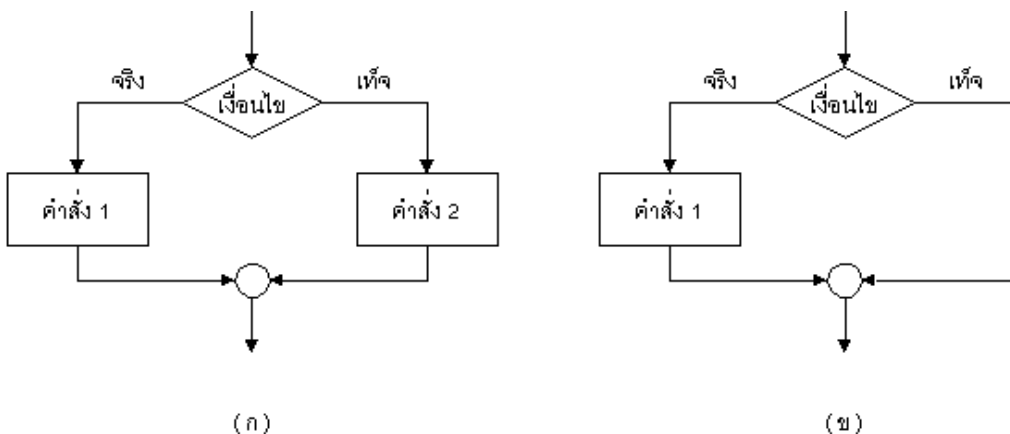
## 1. คำสั่ง if

คำสั่ง if เป็นคำสั่งที่ใช้ในการเขียนแบบเงื่อนไข ตัวอย่างของประโยคในลักษณะเงื่อนไขเป็นตัวอย่างที่สามารถพบเห็นได้ในชีวิตประจำวัน เช่น

ถ้าวันนี้ฝนไม่ตก ฉันจะเดินไปโรงเรียน แต่ถ้าฝนตก ฉันจะขอให้คุณพ่อไปส่งที่โรงเรียน หรือตัวอย่างเช่น

ถ้าฉันสอบได้คะแนนดี คุณพ่อและคุณแม่จะภูมิใจ

จะเห็นว่าประโยคเงื่อนไขดังกล่าวมีอยู่ 2 ลักษณะ คือ ถ้าเงื่อนไขเป็นจริงเกิดเหตุการณ์หนึ่ง แต่ถ้าไม่จริงจะเกิดอีกเหตุการณ์หนึ่ง กับประโยคในลักษณะที่ถ้าเงื่อนไขเป็นจริงจึงจะเกิดเหตุการณ์ขึ้นเท่านั้น ทั้ง 2 ลักษณะสามารถเขียนเป็นผังงานของงานได้ดังรูปที่ 3.1 (ก) และ (ข)



รูปที่ 3.1 แสดงผังงานของประโยคเงื่อนไข

จากผังงานทั้ง 2 จะมีรูปแบบการเขียนคำสั่ง if เกิดขึ้น 2 แบบ

### 1.1 คำสั่ง if-else

คำสั่ง if ในรูปแบบแรกจะคำสั่งที่ต้องทำทั้งในกรณีที่เงื่อนไขเป็นจริงและเป็นเท็จ โดยใช้ นิพจน์ตรรกศาสตร์มาเป็นเครื่องมือช่วยในการตรวจสอบเงื่อนไข มีรูปแบบคำสั่ง คือ

```
if (เงื่อนไข)
 คำสั่งที่ 1;
else
 คำสั่งที่ 2;
```

ตัวอย่างเช่น หากรับข้อมูลจากผู้ใช้และต้องการตรวจสอบว่าเลขที่รับเข้ามามีค่ามากกว่า 10 ให้พิมพ์ข้อความว่า “Number XXXX is over than 10” แต่ถ้าไม่ใช่ให้พิมพ์ข้อความว่า “Number XXXX is not over than 10” จะเขียนเป็นคำสั่งได้ว่า

```
scanf("%d", &number);
if (number > 10)
 printf("Number %d is over than 10", number);
else
 printf("Number %d is not over than 10", number);
```

แต่หากเงื่อนไขเป็นจริงหรือเท็จแล้วต้องทำคำสั่งมากกว่า 1 คำสั่ง จะต้องใช้เขียน if-else ในรูปแบบที่ใช้เครื่องหมาย { } ซึ่งแสดงขอบเขตของการทำเงื่อนไข ครอบคำสั่งที่ต้องทำในแต่ละเงื่อนไข มีรูปแบบดังนี้

```
if (เงื่อนไข) {
 คำสั่งที่ 1;
 คำสั่งที่ 2;

} else {
 คำสั่งที่ 3;
 คำสั่งที่ 4;
 ...
}
```

ตัวอย่างเหมือนในตัวอย่างก่อนหน้านี้ แต่เพิ่มเงื่อนไขว่า ถ้าเลขนั้นมีค่ามากกว่า 10 ให้ลดเลขนั้นลง 5 แสดงได้ดังตัวอย่าง

```
scanf("%d", &number);
if (number > 10) {
 printf("Number %d is over than 10", number);
 number = number - 5;
} else
 printf("Number %d is not over than 10", number);
```

.....

แต่หากมีเงื่อนไขเพิ่มขึ้นอีกว่า ถ้าเลขนั้นไม่มากกว่า 10 ให้เพิ่มค่าเลขนั้นขึ้นอีก 5 สามารถเขียนได้ว่า

```
scanf("%d", &number);
if (number > 10) {
 printf("Number %d is over than 10", number);
 number = number - 5;
} else {
 printf("Number %d is not over than 10", number);
 number = number + 5;
}
```

.....

ทั้งนี้หากมีคำสั่งเพียงคำสั่งเดียวในเงื่อนไข ก็สามารถใช้เครื่องหมาย { } ได้เช่นเดียวกัน เช่น

```
scanf("%d", &number);
if (number > 10) {
 printf("Number %d is over than 10", number);
} else {
 printf("Number %d is not over than 10", number);
}
```

แสดงตัวอย่างโปรแกรมดังตัวอย่างที่ 3.1 และตัวอย่างที่ 3.2

**ตัวอย่างที่ 3.1** โปรแกรมเพื่อตรวจสอบความสูงของนักเรียน 2 คน โดยรับข้อมูลความสูงของนักเรียนทั้งสองมาหาว่าความสูงมากที่สุดคือค่าใด

---

```
#include <stdio.h>
void main() {
 float height1, height2, max;
 printf("Enter first student's height (cm.) : ");
 scanf("%f", &height1);
 printf("Enter second student's height (cm.) : ");
 scanf("%f", &height2);
 if (height1 > height2)
 max = height1;
 else
 max = height2;
 printf("Maximum height is : %.2f cm.", max);
}
```

---

#### ผลการทำงานของโปรแกรม

```
Enter first student's height (cm.) : 184.5
Enter second student's height (cm.) : 192.4
Maximum height is : 192.40 cm.
```

---

**ตัวอย่างที่ 3.2** ให้คำนวณค่าดัชนีน้มน้ำหนัก (Body Mass Index : BMI) ซึ่งสามารถคิดได้จากสูตร  $BMI = w / h^2$  โดยที่ w แทนน้ำหนักตัวมีหน่วยเป็นกิโลกรัม และ h แทนความสูงมีหน่วยเป็นเมตร หากค่า BMI อยู่ในช่วง 20-25 ให้ขึ้นข้อความว่า "Normal BMI." แต่หากอยู่นอกช่วงดังกล่าวให้ขึ้นข้อความว่า "Dangerous BMI."

---

```
#include <stdio.h>
void main() {
 float w, h, BMI;
 printf("Enter weight (Kg): ");
 scanf("%f", &w);
 printf("Enter height (M): ");
 scanf("%f", &h);
 BMI = w / (h * h);
 printf("BMI is %.2f", BMI);
 if (BMI >= 20 && BMI <= 25)
 printf("\nNormal BMI.");
}
```

```

else
 printf("\nDangerous BMI.");
}

```

## 1.2 คำสั่ง if

ในกรณีที่ประโยคเงื่อนไขมีการทำงานเฉพาะเงื่อนไขที่เป็นจริงเท่านั้น โดยไม่มีการทำงานใดในเงื่อนไขที่เป็นเท็จ ดังแสดงในรูปที่ 3.1 (ข) สามารถเขียนแทนด้วยคำสั่ง if โดยไม่ต้องใส่คำสั่ง else แสดงดังรูปแบบ

```

if (เงื่อนไข)
 คำสั่งที่ 1;

```

แต่ถ้าเงื่อนไขเป็นจริงแล้วมีการทำคำสั่งมากกว่า 1 คำสั่งขึ้นไป ก็ใช้รูปแบบของเครื่องหมาย { } ซึ่งใช้ในกรณีที่คำสั่งที่ต้องทำในเงื่อนไขและการวนซ้ำมากกว่า 1 คำสั่ง เพื่อแสดงขอบเขตของการทำงานนั้น

ตัวอย่างเช่น ให้รับข้อมูลจำนวนเต็มจากผู้ใช้ หากข้อมูลนั้นมีค่ามากกว่า 60 หรือน้อยกว่า 20 ให้ขึ้นข้อความว่า "Number XXXX is out of range" เขียนได้ดังตัวอย่าง

```

scanf("%d", &number);
if (number < 20 || number > 60)
 printf("Number %d is out of range", number);

```

แสดงตัวอย่างการใช้งานแสดงดังตัวอย่างที่ 3.3 และ 3.4

**ตัวอย่างที่ 3.3** โปรแกรมเพื่อให้ผู้ใช้คาดเดาคำตัวอักษรที่โปรแกรมได้ตั้งไว้ ถ้าผู้ใช้ป้อนข้อมูลตัวอักษรตรงกับตัวอักษรตรงกับสิ่งที่โปรแกรมตั้งไว้จะขึ้นคำว่า "Bingo"

```

#include <stdio.h>
#define ANS 'G'
void main() {
 char ch;
 printf("Enter character (a-z/A-Z) : ");
 scanf("%c", &ch);
 if (ch == ANS)
 printf("Bingo");
}

```

## ผลการทำงานของโปรแกรม

Enter character (a-z/A-Z) : *G*

Bingo

---

**ระวัง** - หากพิมพ์คำสั่ง `ch == ANS` เป็นคำสั่ง `ch = ANS` จะได้ผลลัพธ์การทำงานที่แตกต่างกัน  
 - การเปรียบเทียบ `ch == ANS` คือการสั่งให้เปรียบเทียบว่า `ch == 'G'`

---

**ตัวอย่างที่ 3.4** แสดงโปรแกรมเพื่อรับข้อมูลเลขจำนวนเต็ม 2 จำนวนจากผู้ใช้ หากค่าแรกที่รับมามีค่ามากกว่าค่าหลังให้ขึ้นข้อความว่า "First value more than second value."

---

```
#include <stdio.h>
void main() {
 int a, b;
 printf("Enter A : ");
 scanf("%d", &a);
 printf("Enter B : ");
 scanf("%d", &b);
 if (a > b)
 printf("First value more than second value");
}
```

---

### 1.3 คำสั่ง if แบบซับซ้อน

ในบางกรณีประโยคเงื่อนไขอาจมีความซับซ้อน มีการเปรียบเทียบเงื่อนไขเดียวกันกับหลายค่า เช่น ให้รับข้อมูลชั้นปีของนักศึกษาและให้พิมพ์ข้อความตรงกับชั้นปี กำหนดว่าชั้นปีที่ 1 พิมพ์ว่า "Freshman" ชั้นปีที่ 2 พิมพ์ว่า "Sophomore" ชั้นปีที่ 3 พิมพ์ว่า "Junior" ชั้นปีที่ 4 พิมพ์ว่า "Senior" ชั้นปีอื่น ๆ พิมพ์ว่า "Super"

```
scanf("%d", &year);
if (year == 1)
 printf("Freshman");
else if (year == 2)
 printf("Sophomore");
else if (year == 3)
 printf("Junior");
else if (year == 4)
 printf("Senior");
else
 printf("Super");
```

นอกจากนี้บางเงื่อนไขอาจมีความซับซ้อน เช่น พิจารณาข้อมูลนักศึกษา ให้ตรวจสอบว่าถ้าเป็นนักศึกษาเพศชายมีความสูงตั้งแต่ 180 ซม. ขึ้นไป ให้ขึ้นข้อความแนะนำว่าควรจะสมัครเข้าชมรมบาสเก็ตบอล แต่ถ้าเป็นเพศหญิงมีความสูงตั้งแต่ 170 ซม. ขึ้นไป ให้ขึ้นข้อความแนะนำว่าควรจะสมัครเข้าชมรมวอลเลย์บอล

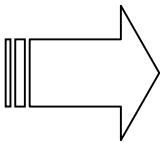
```
if (gender == 'M' && height > 180)
 printf("Basketball");
else if (gender == 'F' && height > 170)
 printf("Volleyball");
```

สามารถเขียนในอีกลักษณะหนึ่งได้ว่า

```
if (gender == 'M') {
 if (height > 180)
 printf("Basketball");
} else { /* ไม่ใช่เพศชาย เพราะฉะนั้นเป็นเพศหญิง */
 if (height > 170)
 printf("Volleyball");
}
```

การเขียนในลักษณะดังกล่าวจะต้องระมัดระวังเรื่องการใช้เครื่องหมาย { } ให้ถูกต้อง

พิจารณาจากตัวอย่างต่อไปนี้

|                                                                                                                                                  |                                                                                                      |                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>if (gender == 'M')     if (height &gt; 180)         printf("Basketball"); else     if (height &gt; 170)         printf("Volleyball");</pre> | <p>เหมือนกับ</p>  | <pre>if (gender == 'M')     if (height &gt; 180)         printf("Basketball"); else if (height &gt; 170)     printf("Volleyball");</pre> |
|--------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|

จะเห็นว่า การเว้นย่อหน้าไม่ช่วยให้เกี่ยวข้องกับการจับคู่ของเงื่อนไข if-else เวลาที่คอมไพเลอร์มองดูโปรแกรมไม่ได้ดูจากการย่อหน้า แต่ดูจากสัญลักษณ์ต่าง ๆ การตีความจึงแตกต่างจากความตั้งใจของผู้เขียนโปรแกรม เพราะฉะนั้นหากไม่แน่ใจการจับคู่เงื่อนไขใด ให้ใช้เครื่องหมาย { } เป็นตัวบอกรอบเขตการทำงานของคำสั่ง if-else นั้น ๆ โดยที่จำนวนปีกกาเปิดจะในโปรแกรมจะต้องเท่ากับจำนวนปีกกาปิดในโปรแกรมเสมอ แสดงตัวอย่างโปรแกรมดังตัวอย่างที่ 3.5 และ 3.6

**ตัวอย่างที่ 3.5** เขียนโปรแกรมเพื่อรับข้อมูลคะแนนสอบของนักศึกษา และให้พิมพ์เกรดที่นักศึกษาได้รับจากเงื่อนไขการให้ลำดับชั้นดังนี้

คะแนนต่ำกว่า 50 ได้เกรด F  
 คะแนนต่ำกว่า 60 ได้เกรด D  
 คะแนนต่ำกว่า 70 ได้เกรด C  
 คะแนนต่ำกว่า 85 ได้เกรด B  
 คะแนนตั้งแต่ 85 ขึ้นไปได้เกรด A

---

```
#include <stdio.h>
void main() {
 float score;
 printf("Enter score : ");
 scanf("%f", &score);
 if (score < 50)
 printf("Grade F");
 else if (score < 60)
 printf("Grade D");
 else if (score < 70)
 printf("Grade C");
 else if (score < 85)
 printf("Grade B");
 else
 printf("Grade A");
}
```

---

**ตัวอย่างที่ 3.6** ให้รับข้อมูลจำนวน 3 จำนวนจากผู้ใช้ และให้หาค่ามากที่สุดมีค่าเท่าใด

---

```
#include <stdio.h>

void main() {
 int first, second, third, max;
 printf("Enter first number : ");
 scanf("%d", &first);
 printf("Enter second number : ");
 scanf("%d", &second);
 printf("Enter third number : ");
 scanf("%d", &third);
 max = first;
 if (max > second)
 max = second;
```



```

else if (max > third)
 max = third;
printf("Maximum number is %d", max);
}

```

---

ในที่นี้ลองพิจารณาตัวอย่างของ Short Circuit ที่ได้กล่าวถึงในบทที่ 2 กับการใช้งานประโยคเงื่อนไข สิ่งที่ต้องระวังคือ หากใช้เครื่องหมาย && (Logical And) ถ้านิพจน์แรกเป็นเท็จ จะทำให้นิพจน์นั้นเป็นเท็จเสมอ และถ้าเป็นเครื่องหมาย || (Logical Or) หากนิพจน์แรกเป็นจริง จะทำให้ทั้งนิพจน์เป็นจริงเสมอ สิ่งตามมาคือ คอมไพเลอร์จะไม่มีการประมวลผลคำสั่งในนิพจน์ที่คู่กัน เรียกว่า Short-circuit แสดงดังตัวอย่าง

```

int a = 10;
if (a > 5 || a++ > 10)
 a = a * 2;
printf("a = %d", a);

```

จะได้คำตอบว่า  $a = 20$  ทั้งนี้หากสังเกตนิพจน์ที่ 2 ของการเปรียบเทียบจะเห็นว่า ผู้เขียนโปรแกรม ตั้งใจให้มีการเพิ่มค่าของ  $a$  ขึ้นอีก 1 ก่อนที่จะทำงานใด ๆ แต่เนื่องจากนิพจน์แรกเป็นจริง ทำให้สรุปได้ว่าทั้งนิพจน์นี้เป็นจริง จึงไม่มีการประมวลผลนิพจน์ที่ 2 ค่า  $a$  จึงไม่เพิ่มขึ้น ซึ่งอาจจะทำให้ค่าที่ได้ผิดจากความตั้งใจของผู้เขียนโปรแกรม

.....

ตัวอย่างของ Short-circuit ที่ใช้เครื่องหมาย && ได้แก่

```

int a = 10;
if (a < 5 && a++ > 10)
 a = a * 2;
printf("a = %d", a);

```

จะได้คำตอบคือ  $a = 10$  เนื่องจากนิพจน์แรกเป็นเท็จ ทำให้เกิด Short-circuit ไม่มีการประมวลผลนิพจน์ที่ 2 ค่า  $a$  จึงไม่มีการเพิ่มขึ้น

.....

นอกจากนี้หากเปรียบเทียบตัวดำเนินการเงื่อนไขที่ได้กล่าวถึงในบทที่ 2 สามารถเปรียบเทียบได้กับการทำงานประโยคเงื่อนไข if-else ดังตัวอย่าง

$$x = (y < 0) ? -y : y;$$

หากเขียนคำสั่งดังกล่าวเป็นประโยคเงื่อนไข if-else จะได้

```

if (y < 0)
 x = -y;
else
 x = y;

```

## 2. คำสั่ง switch


คำสั่ง switch เป็นคำสั่งที่ใช้ในการเขียนประโยคเงื่อนไข มักจะใช้กับกรณีที่เป็นเงื่อนไข if แบบซับซ้อน ตัวอย่างเช่น ในเรื่องของการตรวจสอบชั้นปีของนักศึกษา และให้พิมพ์ข้อความตามชั้นปีที่กำหนด ดังตัวอย่างที่แสดงในหัวข้อ 3.1.3 จะเห็นว่ามีการตรวจสอบนิพจน์เงื่อนไข คือ ชั้นปีของนักศึกษาในทุกเงื่อนไขเหมือนกัน ประโยคในลักษณะเช่นนี้สามารถใช้คำสั่ง switch มาช่วยในการเขียน เพื่อช่วยให้อ่านเข้าใจได้มากยิ่งขึ้น ทั้งนี้เงื่อนไขที่จะนำมาตรวจสอบในคำสั่ง switch ได้จะต้องมีค่าเป็นเลขจำนวนเต็มหรือตัวอักขระเท่านั้น ไม่สามารถใช้ในการตรวจสอบสตริง หรือข้อมูลที่มีลักษณะเป็นช่วง มีรูปแบบของคำสั่ง switch คือ

```
switch (เงื่อนไข) {
 case ค่าคงที่1 : คำสั่ง1 ;
 case ค่าคงที่2 : คำสั่ง2 ;


 default : คำสั่ง N ;
}
```

การทำงานของคำสั่ง switch จะตรวจสอบเงื่อนไข ว่าตรงกับค่า case ไດ ก็จะไปทำงานที่คำสั่งที่อยู่ใน case นั้น คำสั่งหนึ่งที่มักจะใช้คู่กับคำสั่ง switch คือ คำสั่ง break คำสั่งนี้ใช้ในการบอกให้โปรแกรมหยุดการทำงาน และกระโดดออกจากขอบเขตของ { } ที่ใกล้ที่สุด ซึ่งสามารถใช้คำสั่งนี้ร่วมกับคำสั่งวนซ้ำอื่น ๆ อีกด้วย พิจารณาการทำงานของคำสั่ง switch จากตัวอย่าง

```
int a=2;
switch (a) {
 case 1 : printf("11111\n");
 case 2 : printf("22222\n");
 case 3 : printf("33333\n");
 default : printf("AAAAA\n");
}
```



```
int a=2;
switch (a) {
 case 1 : printf("11111\n");
 break;
 case 2 : printf("22222\n");
 break;
 case 3 : printf("33333\n");
 break;
 default : printf("AAAAA\n");
}
```



**ผลการทำงาน (ก)**

22222

33333

AAAAA

**ผลการทำงาน (ข)**

22222

- จากตัวอย่าง ก เมื่อโปรแกรมตรวจสอบนิพจน์ว่าตัวแปร a มีค่าเท่ากับ 2 จะมาทำงานที่คำสั่ง case 2 พิมพ์ค่า 22222 และทำคำสั่งต่อ ๆ มาได้ผลลัพธ์ดังตัวอย่าง
- ตัวอย่าง ข มีการใช้คำสั่ง break เมื่อมีการตรวจสอบว่าตัวแปร a มีค่าเท่ากับ 2 จะมาทำงานที่คำสั่ง case 2 พิมพ์ค่า 22222 และทำคำสั่ง break ซึ่งจะทำให้การทำงานกระโดดออกจากขอบเขตของเครื่องหมาย { } ที่ใกล้ที่สุด ได้ผลลัพธ์ดังตัวอย่าง

คำสั่ง default ใน switch จะมีค่าเหมือนกับ else ในคำสั่ง if-else ก็คือค่าใด ๆ ก็ตามที่ไม่ใช่ค่าที่กำหนดใน case จะมาทำที่คำสั่ง default ซึ่งคำสั่ง default นี้จะมีหรือไม่มีก็ได้ หากเขียนคำสั่ง switch แทนคำสั่ง if-else ของการพิมพ์ค่าของชั้นปีนักศึกษาจะได้ดังตัวอย่าง

```
scanf("%d", &year);
if (year == 1)
 printf("Freshman");
else if (year == 2)
 printf("Sophomore");
else if (year == 3)
 printf("Junior");
else if (year == 4)
 printf("Senior");
else
 printf("Super");
```

```
scanf("%d", &year);
switch (year) {
 case 1 : printf("Freshman");
 break;
 case 2 : printf("Sophomore");
 break;
 case 3 : printf("Junior");
 break;
 case 4 : printf("Senior");
 break;
 default : printf("Super");
}
```

นอกจากนี้ยังสามารถเขียนคำสั่ง switch ในลักษณะอื่น ๆ แสดงดังตัวอย่างที่ 3.7 และ 3.8

**ตัวอย่างที่ 3.7** เขียนโปรแกรมเพื่อรับข้อมูลตัวอักษรจากผู้ใช้ หากผู้ใช้ป้อนตัวอักษร a, b, x ให้ขึ้นข้อความว่า “Hanaga” ป้อนตัวอักษร u, d, p ให้ขึ้นข้อความว่า “Bingo” ป้อนตัวอักษร g ให้ขึ้นข้อความว่า “Google” ป้อนตัวอักษรอื่น ๆ ให้ขึ้นข้อความว่า “Yappadappadoooo”

---

```
#include <stdio.h>

void main() {
 char ch;
 printf("Enter character : ");
 scanf("%c", &ch);
 switch (ch) {
 case 'a' :
 case 'b' :
 case 'x' : printf("Hanaga");
 break;

 case 'u' :
 case 'd' :
 case 'p' : printf("Bingo");
 break;

 case 'g' : printf("Google");
 break;

 default : printf("Yappadappadoooo");
 }
}
```

---

**ตัวอย่างที่ 3.8** เขียนโปรแกรมเพื่อคำนวณค่าจ้างของพนักงาน โดยกำหนดให้รับข้อมูลจำนวนชั่วโมงทำงานและประเภทพนักงาน ซึ่งพนักงานแต่ละประเภทได้รับค่าจ้างต่อชั่วโมงต่างกันดังนี้

| ประเภทของงาน | อัตราค่าจ้าง / ชั่วโมง |
|--------------|------------------------|
| 0            | 30                     |
| 1            | 40                     |
| 2            | 45                     |

---

```
#include <stdio.h>
#include <conio.h>
void main() {
 float workRate, totalPay;
 int workHours;
```

```
char type;
clrscr();
printf("Enter type (1-3) : ");
scanf("%c", &type);
flushall();
if (type == '1' || type == '2' || type == '3') {
 switch (type) {
 case '1' : workRate = 30.0f;
 break;

 case '2' : workRate = 40.0f;
 break;

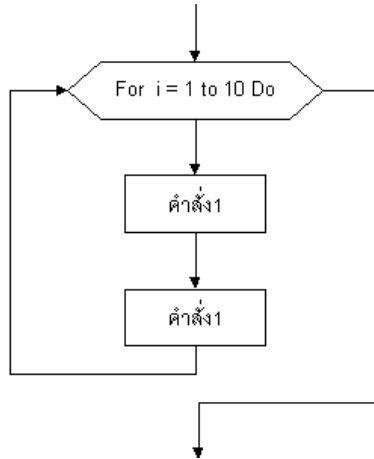
 default : workRate = 45.0f;
 }
 printf("Enter work hours : ");
 scanf("%d", &workHours);
 flushall();
 totalPay = workHours * workRate;
 printf("Rate is %.2f, total pay = %.2f", workRate, totalPay);
} else
 printf("!!!Error : incorrect employee type");
getch();
}
```

---

---

### 3. คำสั่ง for

คำสั่งวนซ้ำเป็นคำสั่งใช้แก้ปัญหาโจทย์ในลักษณะที่มีการทำงานเดิมซ้ำกันหลาย ๆ ครั้ง ซึ่งเขียนในรูปแบบของผังงานได้ดังรูปที่ 3.2



รูปที่ 3.2 แสดงผังงานของคำสั่ง for

คำสั่ง for เป็นคำสั่งวนซ้ำในลักษณะที่รู้จำนวนรอบของการวนซ้ำที่แน่นอน โดยแบ่งรูปแบบหลักออกเป็น 3 ส่วน ได้แก่

- ส่วนที่ใช้กำหนดค่าเริ่มต้นหรือกำหนดค่าตัวนับของการวนซ้ำ
- ส่วนที่ตรวจสอบเงื่อนไขการวนซ้ำ
- ส่วนของการจัดการค่าตัวนับของการวนซ้ำ

```
for (กำหนดค่าตัวนับ ; เงื่อนไขการวนซ้ำ ; จัดการค่าตัวนับ) {
 คำสั่ง1;
 คำสั่ง2;
}
```

ขั้นตอนของการทำงานเมื่อพบคำสั่ง for มีดังนี้

1. ทำคำสั่งในการกำหนดค่าตัวนับ
2. ตรวจสอบเงื่อนไขการวนซ้ำ หากเป็นเท็จจะหยุดและออกจากการทำงานของคำสั่ง for ไปทำงานคำสั่งหลังจากนั้น

3. กรณีเงื่อนไขการวนซ้ำเป็นจริง จะทำคำสั่งในขอบเขตของ for นั้น คือภายใต้เครื่อง { } จนกระทั่งหมด และไปทำคำสั่งจัดการค่าตัวนับ ซึ่งอาจจะเป็นการเพิ่มค่าหรือลดค่าตัวนับ หลังจากนั้นจะกลับทำตรวจสอบเงื่อนไขการวนซ้ำในขั้นตอนที่ 2 ทำเช่นนี้เรื่อยไปจนกระทั่งเงื่อนไขการวนซ้ำเป็นเท็จ

หากคำสั่งที่ต้องทำในการวนซ้ำมีเพียง 1 คำสั่ง รูปแบบการเขียนจะเขียนเครื่องหมาย { } ครอบคำสั่งนั้นไว้หรือไม่ก็ได้ แต่ถ้ามีคำสั่งที่ต้องทำซ้ำมากกว่า 1 คำสั่ง จะต้องมีเครื่องหมาย { } แสดงขอบเขตของการทำวนซ้ำเสมอ พิจารณาตัวอย่างที่ 3.9

**ตัวอย่างที่ 3.9** เขียนโปรแกรมเพื่อรับข้อมูลเลขจำนวนเต็มจากผู้ใช้จำนวน 5 ค่า และหาค่าเฉลี่ยของเลขที่ป้อนเข้ามาเป็นเท่าใด

---

```
#include <stdio.h>

void main() {
 int i, number;
 float average, sum=0.0f;

 for (i=0; i < 5; i++) {
 printf("Enter number %d : ", i+1);
 scanf("%d", &number);
 sum += number;
 }
 average = sum / 5;
 printf("Average is %.2f", average);
}
```

---

#### ผลการทำงานของโปรแกรม

```
Enter number 1 : 10
Enter number 2 : 20
Enter number 3 : 30
Enter number 4 : 40
Enter number 5 : 50
Average is 30.00
```

---

**ระวัง** - หากกำหนดให้ตัวแปร sum เป็นข้อมูลชนิด int การสั่งให้ average = sum / 5; จะทำให้เกิดการปัดเศษทิ้ง

---

ในการทำงานของคำสั่ง for สิ่งสำคัญที่ผู้ใช้จะต้องรู้คือ ทำอย่างไรจึงจะวนทำซ้ำได้เท่ากับจำนวนรอบที่ต้องการ จากตัวอย่างที่ 3.9 มีการใช้ตัวแปร i เป็นตัวนับจำนวนรอบของการทำซ้ำ ในที่นี้ต้องการให้มีการทำซ้ำทั้งหมด 5 รอบ ไล่ขั้นตอนการทำงานของโปรแกรมได้ดังนี้

1. กำหนดให้ i มีค่าเริ่มต้นที่ 0
2. แล้วตรวจสอบว่า i มีค่าน้อยกว่า 5 เงื่อนไขเป็นจริง มีรับข้อมูลตัวแรกและทำการบวกเลขที่รับเข้ามา นั่นเก็บไว้ในตัวแปร sum
3. เพิ่มค่า i ขึ้น 1 เพราะฉะนั้น i มีค่าเท่ากับ 1
4. แล้วตรวจสอบว่า i มีค่าน้อยกว่า 5 เงื่อนไขเป็นจริง มีรับข้อมูลตัวที่ 2 และทำการบวกเลขที่รับเข้ามา นั่นเก็บไว้ในตัวแปร sum
5. เพิ่มค่า i ขึ้น 1 เพราะฉะนั้น i มีค่าเท่ากับ 2
6. แล้วตรวจสอบว่า i มีค่าน้อยกว่า 5 เงื่อนไขเป็นจริง มีรับข้อมูลตัวที่ 3 และทำการบวกเลขที่รับเข้ามา นั่นเก็บไว้ในตัวแปร sum
7. เพิ่มค่า i ขึ้น 1 เพราะฉะนั้น i มีค่าเท่ากับ 3
8. แล้วตรวจสอบว่า i มีค่าน้อยกว่า 5 เงื่อนไขเป็นจริง มีรับข้อมูลตัวที่ 4 และทำการบวกเลขที่รับเข้ามา นั่นเก็บไว้ในตัวแปร sum
9. เพิ่มค่า i ขึ้น 1 เพราะฉะนั้น i มีค่าเท่ากับ 4
10. แล้วตรวจสอบว่า i มีค่าน้อยกว่า 5 เงื่อนไขเป็นจริง มีรับข้อมูลตัวที่ 5 และทำการบวกเลขที่รับเข้ามา นั่นเก็บไว้ในตัวแปร sum
11. เพิ่มค่า i ขึ้น 1 เพราะฉะนั้น i มีค่าเท่ากับ 5
12. ตรวจสอบเงื่อนไข i มีค่าน้อยกว่า 5 พบว่าเงื่อนไขเป็นเท็จ ก็จะจบการทำงานภายในคำสั่ง for

พิจารณาค่าของการวนซ้ำต่อไปนี้ ว่าค่าเริ่มต้นของตัวนับการทำซ้ำเป็นเท่าใด ทำซ้ำทั้งหมดกี่รอบ และค่าสุดท้ายของตัวนับที่ออกจากการทำซ้ำเป็นเท่าใด จากตาราง 3.1

ตาราง 3.1 หาค่าของการทำงานวนซ้ำด้วยคำสั่ง for

| คำสั่ง                       | ค่าตัวนับเริ่มต้น | จำนวนรอบ | ค่าตัวนับสุดท้าย |
|------------------------------|-------------------|----------|------------------|
| for ( i = 1; i <= 5; i++)    | 1                 | 5        | 6                |
| for ( j = 53; j <= 57; j++)  | 53                | 5        | 58               |
| for ( k = 10; k > 5; k- )    | 10                | 5        | 5                |
| for ( a = 2; a < 20; a += 2) | 2                 | 9        | 20               |
| for ( a = 2; a < 20; a *= 2) | 2                 | 4        | 32               |

จะเห็นว่าคำสั่งส่วนต่าง ๆ ของคำสั่ง for สามารถเขียนได้หลายรูปแบบ แต่มีสุดท้ายที่ต้องการคือการควบคุมจำนวนรอบของการทำซ้ำให้ได้เท่ากับจำนวนที่ผู้เขียนโปรแกรมต้องการ



สิ่งที่ต้องระวังในการเขียนคือ

```
for (i = 0; i < 5; i++);
 printf("Hello\n");
```

การใส่เครื่องหมาย ; ต่อท้ายคำสั่ง for เครื่องหมาย ; เป็นคำสั่งที่เรียกว่า Null Statement นับเป็นคำสั่ง 1 คำสั่ง โดยที่คำสั่งนี้จะไม่ทำอะไรเลย เพราะฉะนั้นในตัวอย่างดังกล่าว จะมีการวนทำซ้ำโดยไม่ทำอะไรเลย 5 รอบ แล้วจึงพิมพ์ข้อความ Hello เพียงข้อความเดียว แทนที่จะพิมพ์ข้อความ Hello 5 ครั้ง

นอกจากนี้ที่พบเห็นบ่อย ๆ คือการใช้เครื่องหมาย ; ต่อท้ายคำสั่ง if ซึ่งจะให้ผลเช่นเดียวกับตัวอย่างข้างต้น เช่น

```
if (a > 5);
 printf("Hello");
```

จากการใส่เครื่องหมายดังกล่าวคอมไพเลอร์จะตีความได้ว่า ตรวจสอบว่า a มีค่ามากกว่า 5 หรือไม่ ถ้ามากกว่าก็ไม่ต้องทำอะไร ( ทำ Null Statement คือ ; ) แล้วไปพิมพ์ข้อความว่า Hello แทนที่จะทำการตรวจสอบว่า ถ้า a มีค่ามากกว่า 5 ให้พิมพ์ข้อความว่า Hello

ตัวอย่างการใช้คำสั่ง for เพิ่มเติม คือ คำสั่ง for สามารถใช้ตัวดำเนินการคอมมา ( , ) ประกอบภายในคำสั่ง หรืออาจจะเขียนองค์ประกอบของคำสั่งไม่ครบก็ได้ แสดงดังตัวอย่าง

```
for (i=0, j=0; i<5 && i+j < 10; i++, j+=2) {

}
a = 0;
for (; a < 5 ;){

 a++;
}
```

ตัวอย่างเพิ่มเติมในเรื่องของคำสั่ง for แสดงดังตัวอย่างที่ 3.10 และ 3.11

**ตัวอย่างที่ 3.10** โปรแกรมเพื่อหาค่าเฉลี่ยของเลขจำนวนเต็ม N ตัวซึ่งรับจากผู้ใช้

---

```
#include <stdio.h>
void main() {
 int i, number, n;
 float average, sum=0.0f;
```

```

printf("Enter N : ");
scanf("%d", &n);
for (i=0 ; i < n ; i++) {
 printf("Enter number %d : ", i+1);
 scanf("%d", number);
 sum += number;
}
average = sum / 5;
printf("Average is %.2f", average);
}

```

**ตัวอย่างที่ 3.11** โปรแกรมเพื่อแสดงผลพีธของการแปลงเลขฐาน 10 ให้เป็นเลขฐาน 8 และเลขฐาน 16 โดยรับข้อมูลค่าเริ่มต้นและค่าสุดท้ายของช่วงข้อมูลที่ต้องการแปลงค่า

```

#include <stdio.h>
void main() {
 int start, end, i;
 printf("Enter start number : ");
 scanf("%d", &start);
 printf("Enter end number : ");
 scanf("%d", &end);
 printf("\n\tDec\t\t\tOctal\t\t\tHexa");
 for (i=start ; i <= end ; i++) {
 printf("\n\t%3d\t\t\t%5o\t\t\t%4x", i, i, i);
 }
}

```

นอกจากนี้ยังมีคำสั่งเพิ่มเติมที่จะพบได้เมื่อมีการใช้การทำงานวนซ้ำในลักษณะต่างทั้งคำสั่ง for คำสั่ง while และคำสั่ง do-while คำสั่งดังกล่าวได้แก่ คำสั่ง continue คำสั่งนี้จะทำงานคล้ายกับคำสั่ง break ซึ่งเป็นการบอกให้จบการทำงานในขอบเขตที่คำสั่งอยู่ และออกจากคำสั่งวนซ้ำนั้น คำสั่ง continue จะบอกให้จบการทำงานในขอบเขตที่คำสั่งอยู่เช่นเดียวกัน แต่จะวนกลับไปยังทำคำสั่งวนซ้ำนั้นในรอบถัดไปแทน พิจารณาจากตัวอย่าง โดยที่ให้มีการรับข้อมูลเลขจำนวนเต็ม N จำนวนจากผู้ใช้ ให้หาผลรวมของเลขคู่ที่รับเข้ามา แต่หากผลรวมนั้นมาค่ามากกว่า 200 ให้หยุดการหาผลรวม ถ้าเมื่อใดบอเลข 0 เข้ามาให้หยุดการรับข้อมูลทันที จะได้ว่า

```

printf("Enter N : ");
scanf("%d", &n);
sum = 0;
for (i=0; i < N; i++) {
 printf("Enter number %d : ", i);
 scanf("%d", &number);
 if (number == 0)
 break;
 else if (number % 2 == 0) {
 if (sum > 200)
 continue;
 sum += number;
 }
}

```

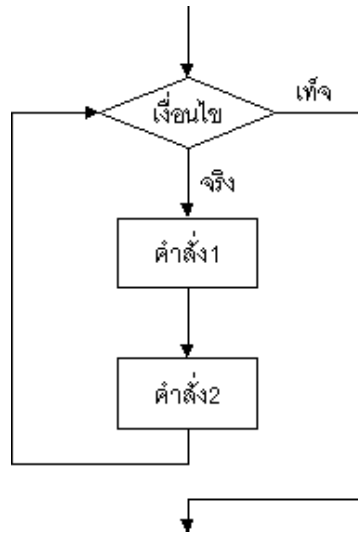
เมื่อใดที่ค่า sum มีค่ามากกว่า 200 จะมีการทำคำสั่ง continue ซึ่งมีผลทำให้ไม่มีการประมวลผลคำสั่งที่เหลือ แต่จะกลับไปทำคำสั่ง i++ และตรวจสอบเงื่อนไขของ  $i < N$  ถ้าเงื่อนไขเป็นจริงก็จะวนทำงานซ้ำต่อไป

#### 4. คำสั่ง while

คำสั่ง while เป็นคำสั่งวนซ้ำ มักใช้ในกรณีที่ต้องทำงานซ้ำกันหลาย ๆ ครั้ง โดยไม่ทราบจำนวนรอบของการทำซ้ำที่แน่นอน ตัวอย่างเช่น

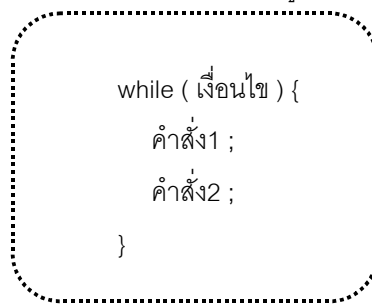
ต้องการรับข้อมูลเลขจำนวนเต็มบวกจากผู้ใช้จำนวนหนึ่งเพื่อนำมาหาค่าเฉลี่ยของตัวเลขที่ป้อนเข้ามาทั้งหมด

การทำงานดังกล่าวจะต้องมีการทำงานวนซ้ำเพื่อรับข้อมูลและหาผลรวมของข้อมูลที่รับเข้ามานั้น ในกรณีเช่นนี้ผู้ที่ใช้งานโปรแกรมซึ่งป้อนจำนวนข้อมูลเข้าสู่ระบบอาจจะป้อนข้อมูลในจำนวนที่ไม่เท่ากัน หากต้องการเขียนโปรแกรมเพื่อให้ทำงานกับผู้ใช้คนใด ๆ มักจะใช้คำสั่ง while เข้ามาช่วยในการเขียนโปรแกรม ในกรณีตัวอย่างเราทราบว่าข้อมูลที่รับเข้ามาต้องเป็นข้อมูลจำนวนเต็มบวกเท่านั้นจึงจะนำมาหาค่าเฉลี่ย ซึ่งผู้เขียนโปรแกรมสามารถตั้งเงื่อนไขว่า หากมีการป้อนข้อมูลเป็นเลขจำนวนเต็มลบให้แสดงว่าผู้ใช้ต้องการหยุดการป้อนข้อมูลนั้น การทำงานของคำสั่ง while สามารถเขียนแสดงด้วยผังงานดังรูปที่ 3.3



รูปที่ 3.3 ผังงานของการทำงานคำสั่ง while

หากแทนผังงานดังกล่าวด้วยคำสั่ง while สามารถเขียนรูปแบบของคำสั่ง while ได้ดังนี้



จากรูปแบบคำสั่งดังกล่าว จะเกิดการทำงานคำสั่ง1 และคำสั่ง2 เมื่อมีการตรวจสอบว่าเงื่อนไขเป็นจริง และจะทำงานซ้ำเช่นนี้ไปจนกว่าเงื่อนไขนั้นจะเป็นเท็จ หากคำสั่งที่ทำซ้ำมีมากกว่า 1 คำสั่งจะต้องใช้เครื่องหมายแสดงขอบเขต คือ { } ครอบคำสั่งที่ต้องการให้ทำซ้ำทั้งหมด แต่ถ้ามีคำสั่งที่ต้องทำซ้ำเพียงคำสั่งเดียว ผู้เขียนโปรแกรมใส่เครื่องหมาย { } หรือไม่ก็ได้

ทั้งนี้ต้องระวังห้ามใส่เครื่องหมาย ; หลังวงเล็บของเงื่อนไข ซึ่งถือเป็นคำสั่ง Null Statement ดังอธิบายในหัวข้อ 3.4 แสดงตัวอย่างโปรแกรมด้วยตัวอย่างที่ 3.12

**ตัวอย่างที่ 3.12** โปรแกรมเพื่อหาค่าเฉลี่ยของเลขจำนวนเต็มบวกที่ผู้ใช้ป้อนเข้าสู่ระบบ เมื่อใดที่ผู้ใช้ป้อนเลขจำนวนเต็มลบให้ถือว่าสิ้นสุดการป้อนข้อมูล

```

#include <stdio.h>

void main() {
 int number, count;
 float sum, average;

```

```

count = 0;
sum = 0.0f;
printf("Enter number : ");
scanf("%d", &number);
while (number >= 0) {
 sum += number;
 count++;
 printf("Enter number : ");
 scanf("%d", &number);
}
average = sum / count;
printf("Average is %.2f", average);
}

```

จากตัวอย่างเนื่องจากจะต้องหาค่าเฉลี่ยของเลขจำนวนเต็มบวกที่ป้อนเข้ามาจึงใช้ตัวแปร count ในการนับจำนวนตัวเลขจำนวนเต็มบวกที่ป้อนเข้ามาทั้งหมด และมีเงื่อนไขในการทำซ้ำคือ number >= 0 ถ้าตรวจใดที่ผู้ใช้ป้อนข้อมูลเข้ามามีค่ามากกว่าหรือเท่ากับ 0 ก็จะทำให้เกิดการหาค่าผลรวมของข้อมูลนั้น และนับจำนวนข้อมูลที่ได้รับเข้ามา ตลอดจนรับข้อมูลตัวใหม่เข้าตรวจสอบว่าตรงกับเงื่อนไขของการทำซ้ำหรือไม่ จนกว่าจะมีการป้อนข้อมูลน้อยกว่า 0 จึงจะมีการคำนวณหาค่าเฉลี่ย และแสดงผลลัพธ์ทางจอภาพ

หากสังเกตรูปแบบการเขียนตัวอย่างดังกล่าว จะเขียนเป็นโครงร่างได้ดังนี้

```

printf("Enter number : ");
scanf("%d", &number);
while (number >= 0) {

 printf("Enter number : ");
 scanf("%d", &number);
}

```

} กำหนดค่าเริ่มต้นให้ค่าควบคุม  
 → เงื่อนไขค่าควบคุมการทำซ้ำ  
 } เปลี่ยนแปลงค่าควบคุม

ในการทำงานของคำสั่ง while จะมีตัวควบคุมการทำซ้ำเสมอ ซึ่งจะต้องมีการกำหนดค่าเริ่มต้นของค่าควบคุม ตรวจสอบเงื่อนไขค่าควบคุม และมีการเปลี่ยนแปลงค่าควบคุม เพื่อนำไปตรวจสอบเงื่อนไขการทำซ้ำจากขั้นตอนทั้ง 3 พิจารณาเปรียบเทียบกับคำสั่ง for ดังตัวอย่าง

```

for (i = 0; i < 10; i++) {

}

```

ในคำสั่ง for ดังตัวอย่างมี i เป็นค่าควบคุม มีเงื่อนไขตรวจสอบค่าควบคุม และมีการเปลี่ยนแปลงค่าควบคุมเพื่อนำไปสู่การตรวจสอบเงื่อนไขอีกครั้ง ซึ่งสามารถเขียนในรูปแบบของคำสั่ง while ได้ว่า

```

i = 0;
while (i < 10) {
 ...
 i++;
}

```

จะเห็นว่าคำสั่ง for นั้นสามารถเขียนในรูปแบบของคำสั่ง while ได้เช่นเดียวกัน ต่างกันที่คำสั่ง for มักจะใช้กับการทำงานที่รู้จำนวนรอบของการทำซ้ำที่แน่นอน ส่วนคำสั่ง while สามารถใช้ได้ทั่วไปแต่จะเหมาะกับการทำงานที่ไม่รู้จำนวนรอบการทำซ้ำดังตัวอย่างข้างต้น ตัวอย่างเพิ่มเติมแสดงดังตัวอย่างที่ 3.13

**ตัวอย่างที่ 3.13** รับข้อมูลจำนวนเต็มจากผู้ใช้ จนกว่าผู้ใช้จะป้อนเลข -9999 ให้หาว่ามีเลขจำนวนเต็มลบ (ไม่รวม -9999) อยู่กี่จำนวน และมีเลขจำนวนเต็มบวก (รวมทั้งเลขศูนย์) อยู่กี่จำนวน

---

```

#include <stdio.h>

void main() {
 int countPlus, countMinus, num;
 countPlus = countMinus = 0;
 printf("Enter number (-9999 for end) : ");
 scanf("%d", &num);
 while (num != -9999) {
 if (num < 0)
 countMinus++;
 else
 countPlus++;
 printf("Enter number (-9999 for end) : ");
 scanf("%d", &num);
 }
 printf("Number less than zero = %d, more than or equal to zero = %d",
 countMinus, countPlus);
}

```

---

ในกรณีของการใช้คำสั่งวนซ้ำต่าง ๆ สิ่งที่ต้องระวังคือการนำเอาเลขจำนวนจริงมาเป็นค่าควบคุมการวนซ้ำ พิจารณาจากตัวอย่าง เนื่องจากเรารู้ว่า  $1/3 + 1/3 + 1/3$  มีค่าเท่ากับ 1 เราจึงจะใช้วิธีการดังกล่าวมาเป็นตัวควบคุมการทำงานวนซ้ำ เช่น

```
float x, y;
x = 1.0f;
y = 1/3;
while (x != y) {
 printf("\n55555");
 y += 1/3;
}
```

หากทดลองเรียกใช้โปรแกรมดังกล่าว จะเกิดการพิมพ์ค่า 55555 ไปเรื่อย ๆ ทำให้เกิดสิ่งๆที่เรียกว่า การวนซ้ำแบบอนันต์ หรือการวนซ้ำแบบไม่รู้จบ ทั้งนี้เกิดจากการหาของภาษาซี 1 และ 3 เป็นจำนวนเต็ม เมื่อนำมาหารจะได้ค่าเท่ากับ 0 สามารถทดลองง่าย ๆ ด้วยคำสั่งเงื่อนไข

```
x = 1.0f;
y = (1/3) + (1/3) + (1/3);
if (x == y)
 printf("Equal");
else
 printf("Not equal");
```

คำตอบที่ได้คือ Not equal เพราะฉะนั้นหากใช้เลขจำนวนจริงและการคำนวณเป็นเป็นสิ่งควบคุม การวนซ้ำหรือนำไปเปรียบเทียบเงื่อนไขจะต้องระมัดระวังในการเขียนโปรแกรม แสดงตัวอย่างเพิ่มเติมในตัวอย่างที่ 3.14 และ 3.15

**ตัวอย่างที่ 3.14** เขียนโปรแกรมเพื่อรับค่าเลขจำนวนจริงจากผู้ใช้ 1 จำนวน และให้ลบออกด้วย 0.16523 อยากทราบว่าจะต้องลบออกกี่ครั้งเลขที่รับเข้ามานั้นจึงจะมีค่าเข้าใกล้ 0 มากที่สุด (เข้าใกล้ทางบวก) และค่าที่เข้าใกล้ 0 นั้นมีค่าเท่าใด

---

```
#include <stdio.h>
#define MINUS_VALUE 0.16523
void main() {
 int count=0;
 float num, ;
 printf("Enter number : ");
 scanf("%f", &num);
 while (num > 0) {
 num -= MINUS_VALUE;
 count++;
 }
 printf("It's need %d times, value is %f", count-1, num+MINUS_VALUE);
}
```

---

จากตัวอย่างมีการใช้คำสั่ง #define เป็นการบอกให้รู้ว่าหากเมื่อใดเจอข้อความที่อยู่ตรงกลาง จะกำหนดให้มีค่าเท่ากับค่าที่อยู่ทางขวาสุด เช่น #define MINUS\_VALUE 0.16523 แปลได้ว่าเมื่อใดที่คำสั่งในโปรแกรมมีการอ้างถึงคำว่า MINUS\_VALUE ให้แทนที่ข้อความนั้นด้วย 0.16523 เพราะฉะนั้นเมื่อพบคำสั่ง

```
num -= MINUS_VALUE;
```

จะมีค่าเท่ากับคำสั่ง

```
num -= 0.16523;
```

ในตัวอย่างที่ 3.14 เมื่อต้องการพิมพ์ทราบคำตอบจะต้องมีการเพิ่มค่า num และลดค่า count เนื่องจากในมีการทำวนซ้ำ ตรวจสอบไว้ว่า  $num > 0$  ตรวจสอบดูที่ค่า num มีค่ามากกว่า 0 จะทำงานไปเรื่อย ๆ และจะจบก็ต่อเมื่อค่า num มีค่ามากกว่าหรือเท่ากับ 0 แต่คำตอบต้องการค่า num ที่เข้าใกล้ 0 ทางบวก เพราะฉะนั้นจึงต้องมีการเพิ่มค่าให้ num เป็นสถานะก่อนสุดท้าย คือ บวกด้วย MINUS\_VALUE และลดค่า count ลง 1

### ตัวอย่างที่ 3.15 โปรแกรมเพื่อรับคะแนนสอบของนักเรียนให้อยู่ในช่วง 10 ถึง 20

---

```
#include <stdio.h>

#define YES 1
#define NO 0

void main() {
 int num, correct=NO;

 printf("Enter number : ");
 scanf("%d", &num);
 while (!correct) {
 if (num >= 10 && num <= 20)
 correct = YES;
 else {
 printf("Error data !!!\n");
 printf("Enter number : ");
 scanf("%d", &num);
 }
 }
}
```

---

ในตัวอย่างที่ 3.15 สามารถนำไปประยุกต์ใช้กับการตรวจสอบค่าของข้อมูลที่ได้รับเข้ามาให้อยู่ในช่วงที่กำหนดไว้ ในที่นี้ใช้ตัวแปร correct เป็นตัวควบคุมการวนซ้ำ โดยกำหนดเริ่มต้นให้เท่ากับ NO คือ 0 เมื่อตรวจสอบเงื่อนไข !correct จะได้ค่า 1 ทำให้การวนซ้ำเป็นจริง จะเข้าไปทำคำสั่งใน while หากตรวจสอบได้ว่า



ข้อมูลอยู่ในช่วงที่กำหนดจริง จะกำหนดให้ค่า correct = YES หรือ 1 และกลับไปตรวจสอบเงื่อนไข จะได้ว่า !correct ในรอบนี้คือ !1 ได้ค่าเท่ากับ 0 ทำให้การวนซ้ำเป็นเท็จ แต่ถ้าข้อมูลอยู่นอกช่วงที่กำหนดดังกล่าว ก็ จะขึ้นข้อความแสดงความผิดพลาดและกลับไปรับข้อมูลใหม่ สามารถเขียนการตรวจสอบเงื่อนไขอีกแบบหนึ่ง ได้ว่า

```
printf("Enter number : ");
scanf("%d", &num);
while (num < 10 || num > 20) {
 printf("Error data !!!\n");
 printf("Enter number : ");
 scanf("%d", &num);
}
```

ในการเขียนโปรแกรมคำสั่งอาจจะประกอบขึ้นมาจากคำสั่งต่าง ๆ หลายคำสั่งทำงานด้วยกัน ทั้งคำสั่ง ทัวไป คำสั่งเงื่อนไข และคำสั่งวนซ้ำ พิจารณาตัวอย่างเพิ่มเติมของการใช้คำสั่งร่วมกันหลายรูปแบบ จากตัว อย่างที่ 3.16

**ตัวอย่างที่ 3.16** โปรแกรมเพื่อรับข้อมูลจากผู้ใช้ N คน โดยที่ผู้ใช้แต่ละคนป้อนเลขจำนวนเต็มเข้าสู่ ระบบได้ตามความพอใจจนกว่าจะป้อนเลข 9999 ให้หาค่าเฉลี่ยของเลขที่ผู้ใช้แต่ละคนป้อน และหาค่าเฉลี่ย ของเลขที่ป้อนเข้าสู่ระบบทั้งหมด และหาว่าผู้ใช้แต่ละคนป้อนเลขจำนวนเต็มที่อยู่ในช่วง 0 ถึง 30 คนละกี่ตัว และหาว่ามีกรป้อนเลขเลขจำนวนเต็มลบที่อยู่ในช่วง -1 ถึง -30 กี่ตัวโดยผู้ใช้ทั้งหมดกี่ตัว

---

```
#include <stdio.h>
#include <conio.h>

void main() {
 int n, num, countOne, countAll, countSpecOne, countSpecAll, i;
 float sumOne, sumAll, averageOne, averageAll;

 clrscr();
 countAll = countSpecAll = 0;
 sumAll = 0.0f;
 printf("Enter N : ");
 scanf("%d", &n);
 for (i = 1; i <= n; i++) {
 countOne = countSpecOne = 0;
 sumOne = 0.0f;
 printf("\n\nFor user no. %d\n", i);
 printf("Enter number : ");
 scanf("%d", &num);
```

```

while (num != 9999) {
 countOne++;
 sumOne = sumOne + num;
 if (num >= 0 && num <= 30)
 countSpecOne++;
 else if (num <= -1 && num >= -30)
 countSpecAll++;
 printf("Enter number : ");
 scanf("%d", &num);
}
averageOne = sumOne / countOne;
printf("Average for user no. %d is %.2f", i, averageOne);
printf("\nNumber of data between 0 and 30 is %d", countSpecOne);
countAll = countAll + countOne;
sumAll = sumAll + sumOne;
}
averageAll = sumAll / countAll;
printf("\n\nAverage for all user is %.2f", averageAll);
printf("\nNumber of data between -1 and -30 is %d", countSpecAll);
getch();
}

```

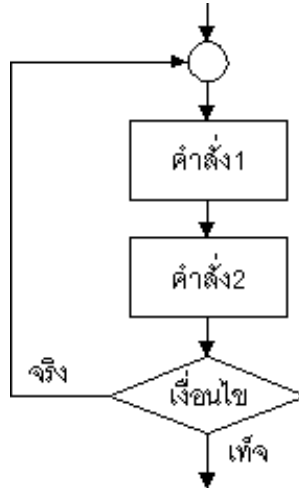
จากตัวอย่างสิ่งที่ต้องระวังคือการหาค่าตัวนับต่าง ๆ ว่าเป็นตัวนับสำหรับงานใด และสำหรับผู้ใช้แต่ละคนหรือว่าสำหรับข้อมูลทั้งหมด หากโปรแกรมซับซ้อนควรจะเริ่มจากการแก้ปัญหาย่อย ๆ ทีละข้อ โดยอาจจะแยกเป็นขั้นตอนคือ

1. รับข้อมูลค่า N คือ จำนวนผู้ใช้ก่อน สร้างการวนซ้ำสำหรับการทำงานของผู้ใช้ N คน อาจจะใช้คำสั่ง for หรือ คำสั่ง while ก็ได้
2. พิจารณาการทำงานของผู้ใช้แต่ละคน รับข้อมูลของผู้ใช้แต่ละคน ซึ่งต้องรับข้อมูลจนกว่าจะป้อนเลข 9999 จึงจะหยุดการทำงาน ซึ่งสามารถใช้คำสั่ง while มาช่วยในการทำงาน
3. แก้ปัญหาที่โจทย์ต้องการทีละอย่าง โดยอาจจะเริ่มจากหาค่าเฉลี่ยของผู้ใช้แต่ละคน
4. หาค่าเฉลี่ยของผู้ใช้ทั้งหมด
5. หาค่าที่อยู่ในช่วง 0 ถึง 30 สำหรับผู้ใช้แต่ละคน
6. หาค่าที่อยู่ในช่วง -1 ถึง 30 สำหรับผู้ใช้ทั้ง

การแบ่งงานเป็นปัญหาย่อยแล้วแก้ปัญหาย่อยเหล่านั้นจะช่วยให้การเขียนโปรแกรมทำได้ง่ายขึ้น ซึ่งเป็นหลักการของการแบ่งแยกและเอาชนะ (Divide and Conquer) ซึ่งใช้ทั่วไปในงานเขียนโปรแกรม และสามารถเขียนเป็นงานย่อยซึ่งจะกล่าวถึงในบทที่ 4 เรื่องของฟังก์ชันต่อไป

## 5. คำสั่ง do-while

คำสั่ง do-while เป็นคำสั่งวนซ้ำอีกรูปแบบหนึ่ง แสดงผังงานดังรูปที่ 3.4



รูปที่ 3.4 แสดงผังงานของการทำงานคำสั่ง do-while

จากรูปจะเห็นว่าการทำงานของคำสั่ง do-while จะต้องมีการทำงานคำสั่ง 1 และคำสั่ง 2 เสมอ หลังจากนั้นจะมีการตรวจสอบเงื่อนไข หากเงื่อนไขเป็นจริงก็จะกลับไปทำคำสั่งใน do-while อีก จนกว่าเงื่อนไขนั้นจะเป็นเท็จ เขียนในรูปแบบของคำสั่งได้ว่า

```
do {
 คำสั่ง 1 ;
 คำสั่ง 2 ;
} while (เงื่อนไข) ;
```

คำสั่ง do-while จะต่างกับคำสั่ง while ตรงที่จะมีการทำงานคำสั่งก่อนตรวจสอบเงื่อนไข ในขณะที่คำสั่ง while จะมีการตรวจสอบเงื่อนไขก่อนการทำงานเสมอ พิจารณาจากตัวอย่างการรับข้อมูลให้อยู่ในช่วงที่ต้องการ สมมติให้อยู่ในช่วงตั้งแต่ 10 ถึง 20 ซึ่งเป็นที่แน่นอนว่าจะต้องมีการรับข้อมูลเข้ามาก่อน จากนั้นจะตรวจสอบว่าข้อมูลถูกต้องหรือไม่ หากเขียนในรูปแบบของคำสั่ง do-while จะได้ว่า

```
do {
 printf("Enter number (between 10 and 20) : ");
 scanf("%d", &num);
} while (num < 10 || num > 20);
```

จากตัวอย่างจะมีการรับข้อมูล แล้วตรวจสอบว่าข้อมูลอยู่นอกช่วงที่กำหนด คือ น้อยกว่า 10 หรือมากกว่า 20 ถือว่าข้อมูลไม่ถูกต้องให้ไปรับข้อมูลใหม่จนกว่าจะถูกต้อง พิจารณาตัวอย่างที่ 3.17 และ 3.18

**ตัวอย่างที่ 3.17** รับข้อมูลจำนวนเต็มจากผู้ใช้ และหาค่าเฉลี่ยของข้อมูลที่ได้รับเข้ามา จนกว่าผู้ใช้จะป้อนเลข 9999

---

```
#include <stdio.h>
void main() {
 int num, count=0;
 float sum=0.0f, average;
 do {
 printf("Enter number : ");
 scanf("%d", &num);
 if (num != 9999) {
 count++;
 sum += num;
 }
 } while (num != 9999);
 average = sum / count;
 printf("Average is %.2f", average);
}
```

---

**ตัวอย่างที่ 3.18** รับข้อมูลจำนวนเต็มจากผู้ใช้ จะหยุดรับข้อมูลเมื่อผลรวมของข้อมูลนั้นมีค่ามากกว่า 200 ให้หาว่ามีการรับข้อมูลทั้งหมดกี่จำนวน

---

```
#include <stdio.h>
void main() {
 int num, sum=0, count=0;
 do {
 printf("Enter number : ");
 scanf("%d", &num);
 sum += num;
 count++;
 } while (sum <= 200);
 printf("Enter number %d times", count);
}
```

---

คำสั่ง do-while สามารถนำไปประยุกต์ใช้ได้เหมือนกับคำสั่ง while หรือ for แต่มักจะใช้คำสั่งนี้ในการตรวจสอบความถูกต้องของข้อมูล หรือใช้ในการสั่งให้โปรแกรมทำงานวนซ้ำจนกว่าผู้ใช้ต้องการให้หยุดการ

ทำงาน จากตัวอย่างที่ได้แสดงมาทั้งหมดจะเห็นว่าเป็นการทำงานเพียงครั้งเดียวก็จบการทำงานของโปรแกรม หากต้องการทำงานนั้นอีกครั้ง ก็จะต้องเรียกใช้งานโปรแกรมนั้นใหม่ สามารถใช้คำสั่ง do-while มาช่วยได้ดังตัวอย่างที่ 3.19

**ตัวอย่างที่ 3.19** รับข้อมูลจำนวนเต็มจากผู้ใช้ จะหยุดรับข้อมูลเมื่อผลรวมของข้อมูลนั้นมีค่ามากกว่า 200 ให้หาว่ามีการรับข้อมูลทั้งหมดกี่จำนวน เมื่อโปรแกรมทำงานเสร็จให้ทำงานวนรับข้อมูลต่อไป โดยขึ้นข้อความสอบถามว่า Exit program (Y/N) ? หากผู้ใช้ป้อนข้อมูล y หรือ Y ให้หยุดการทำงานของโปรแกรม แต่ถ้าป้อนข้อมูลอื่น ๆ ให้เรียกโปรแกรมเดิมกลับมาทำงานอีกครั้ง

---

```
#include <stdio.h>

void main() {
 int num, sum=0, count=0;
 char ch;
 do {
 do {
 printf("Enter number : ");
 scanf("%d", &num); fflush();
 sum += num;
 count++;
 } while (sum <= 200);
 printf("Enter number %d times", count);

 printf("\n\nExit program (Y/N) ? ");
 scanf("%c", &ch);
 } while (ch == 'y' || ch == 'Y');
}
```

---

**แบบฝึกหัดบทที่ 3**

1. หาที่ผิดของคำสั่งต่อไปนี้และแก้ไขให้ถูกต้อง

(ก) IF a = 0 \_\_\_\_\_

(ข) if (a>0 and a<20) or (10<=b<=40)) \_\_\_\_\_

a = a + b \_\_\_\_\_

else c > 50; \_\_\_\_\_

a = a + c \_\_\_\_\_

(ค) if (b = 20 && b <>a); \_\_\_\_\_

a += b; \_\_\_\_\_

b = b + 1; \_\_\_\_\_

if else (b < 10) \_\_\_\_\_

b = b + 2; \_\_\_\_\_

(ง) switch (a=4); (ข้อนี้ให้หาเฉพาะที่ผิดพลาด) \_\_\_\_\_

case 10-20 : a = 10; \_\_\_\_\_

break; \_\_\_\_\_

case > 30 : a = 20; \_\_\_\_\_

case b : a = 30; \_\_\_\_\_

default : a = 40; \_\_\_\_\_

(จ) for (i=5; i=10; i+1); \_\_\_\_\_

(ฉ) while (i=5); \_\_\_\_\_

a = a + 5; \_\_\_\_\_

b = a \* 2; \_\_\_\_\_

(ช) i = 5; \_\_\_\_\_

do \_\_\_\_\_

printf("%d\n" i) \_\_\_\_\_

while (i++ <> 10); \_\_\_\_\_

2. หาผลลัพธ์การทำงานของคำสั่งต่อไปนี้

if (sex == 'M' && age > 40) {

printf("\n11111");

if (age > 60)

printf("\n22222");

} else if (sex == 'F')

printf("\n33333");

else

printf("\n44444");

(ก) เมื่อ sex มีค่า 'M' และ age มีค่า 50

(ข) เมื่อ sex มีค่า 'M' และ age มีค่า 80

(ค) เมื่อ sex มีค่า 'M' และ age มีค่า 25

(ง) เมื่อ sex มีค่า 'F' และ age มีค่า 50

(จ) เมื่อ sex มีค่า 'F' และ age มีค่า 80

(ง) เมื่อ sex มีค่า 'U' และ age มีค่า 30

3. หาจำนวนรอบที่ทำงาน ค่า  $i$  สุดท้ายในลูป และค่า  $i$  เมื่อทำงานจบลูป (จบการทำงานวนรอบ) ต่อไปนี้

|                                 | จำนวนรอบที่ทำงาน | ค่า $i$ สุดท้ายในลูป | ค่า $i$ เมื่อจบลูป |
|---------------------------------|------------------|----------------------|--------------------|
| (ก) for (i=0; i < 10; i++)      | _____.           | _____.               | _____.             |
| (ข) for (i=0; i < 100; i++)     | _____.           | _____.               | _____.             |
| (ค) for (i=50; i > 0; i-=2)     | _____.           | _____.               | _____.             |
| (ง) for (i=10; i+5 < 100; i*=2) | _____.           | _____.               | _____.             |

4. หาผลลัพธ์ของการทำงานต่อไปนี้

|                                                                               |                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (ก) for (i=0; i < 10; i++)<br>printf("\n11111");<br>printf("\n22222");        | (ข) for (i=0; i < 10; i+=2) {<br>printf("\n11111");<br>if (a >= 5)<br>printf("\n22222");<br>}                                                                                                            |
| (ค) for (i=0; i < 10; i++) {<br>printf("\n11111");<br>printf("\n22222");<br>} | (ง) for (i=0, j=5; i < 10 && j < 20; i++, j+=3) {<br>printf("\n11111");<br>if (i > 4 && j > 10)<br>printf("\n22222");<br>else if (i > 8)<br>printf("\n33333");<br>if (j > 15)<br>printf("\n44444");<br>} |

5. แปลงคำสั่ง switch ต่อไปนี้เป็นคำสั่ง if-else และหาผลลัพธ์ของการทำงาน

|                                                                                                                                                               |                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| switch (a) {<br>case 5 : a = 3;<br>break;<br>case 10 : a = 5;<br>case 15 : b = a;<br>break;<br>case 20 : a = 10;<br>break;<br>default : a = b;<br>b = 0;<br>} | (ก) เมื่อ a = 0<br>(ข) เมื่อ a = 5<br>(ค) เมื่อ a = 10<br>(ง) เมื่อ a = 15<br>(จ) เมื่อ a = 18<br>(ฉ) เมื่อ a = 20<br>(ช) เมื่อ a = 40 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|

6. แปลงคำสั่ง if-else ต่อไปนี้เป็นคำสั่ง switch

```

if (a == 0)
 a = 10;
else if (a == 5) {
 x = a + b;
 if (b == 10)
 a = b;
} else if (a == 10 || a == 11 || a == 12) {
 a = a * 2;
 b = b / 2;
} else
 a = 0;

```

7. แปลงคำสั่ง for เป็นคำสั่ง while

```

(ก) for (i = 10; i < 20; i++)
(ข) for (k = 0, m = 10; k < 10; k+=2, m*=2)
(ค) for (j=-10; j < 0 || j+3 < 1; j+=2)
(ง) for (i = 0; i < 50 && i * 2 > 60; i += 4)

```

8. แปลงคำสั่ง while เป็นคำสั่ง for

|                                                                                                 |                                                                            |
|-------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| <pre> (ก) i = 20; while (i &lt; 100 &amp;&amp; i * 2 &lt; 140) {     .....     i *= 2; } </pre> | <pre> (ข) j = k = 0; while (j + k &lt; 20) {     j += 2;     k++; } </pre> |
| _____.                                                                                          | _____.                                                                     |

9. จัดรูปแบบการเยื้อง (Indentation) ให้เหมาะสม

```

(ก) if (a > 0 && b > 0) a = a + 2; else if (a > 50) a = 10; else if (b < 20 || a < -15) { a = b;
 b = b * 2; }else {a = 5; b = 10;}
(ข) switch (x) { case 10 : a = 5; b = 4; break; case 20 : case 30 : case 40 : a = 10; break; case
 50 : a = b; break; default : b = a; a = 12; }
(ค) for(i=0; i < 15; i++) { a = a + b; b = 15; c = a * 2; if (a >= 15) { x = x + 10; } else { y++; c++;
 } d = x + y + c; }

```

10. เขียนโปรแกรมเพื่อรับค่าข้อมูลจำนวนเต็มจากผู้ใช้ ให้หาว่าถ้าเพิ่มค่าข้อมูลนั้นขึ้น 2 เท่าจะมีค่าเท่าใด

11. เขียนโปรแกรมเพื่อคำนวณยอดภาษี VAT โดยคิดภาษี VAT ที่ 7 % ให้รับข้อมูลยอดเงินที่ลูกค้าซื้อสินค้า และแสดงยอดภาษี และยอดเงินที่ลูกค้าต้องจ่ายทั้งหมด



12. เขียนข้อมูลรับข้อมูลเลขจำนวนเต็มจากผู้ใช้ และให้คำนวณว่าเลขดังกล่าวคิดเป็นเวลา กี่ชั่วโมง กี่นาที และกี่วินาที
13. เขียนโปรแกรมเพื่อคำนวณการทอนเงินให้กับลูกค้า โดยรับข้อมูลยอดซื้อของลูกค้า และยอดเงินที่ลูกค้าจ่าย ให้คำนวณว่าต้องทอนเงินให้กับลูกค้าเป็นแบงก์ 1000 500 100 50 20 และเหรียญ 10 5 และ 1 บาทจำนวนอย่างละเท่าใด (กำหนดว่าไม่มีการทอนเงินเป็นเศษสตางค์)
14. เขียนโปรแกรมเพื่อรับข้อมูลตัวอักษรจากผู้ใช้ หากผู้ใช้ป้อนตัวอักษร a, b, x ให้ขึ้นข้อความว่า "Hanaga" ป้อนตัวอักษร u, d, p ให้ขึ้นข้อความว่า "Bingo" ป้อนตัวอักษร g ให้ขึ้นข้อความว่า "Google" ป้อนตัวอักษรอื่น ๆ ให้ขึ้นข้อความว่า "Yappadappadoo" โดยใช้คำสั่ง if-else
15. เขียนโปรแกรมเพื่อรับข้อมูลเลขจำนวนจริง 3 ค่าจากผู้ใช้ และพิมพ์ค่าดังกล่าวเรียงจากมากไปน้อย
16. เขียนโปรแกรมเพื่อตรวจสอบปี ค.ศ. ที่รับจากผู้ใช้เพื่อตรวจสอบว่าเป็นปีอธิกสุรทินหรือไม่ โดยที่ปีอธิกสุรทินเป็นปีที่หารด้วย 4 ลงตัว กรณีที่เป็นปีที่ปีเป็นศตวรรษจะต้องหารด้วย 400 ลงตัว เช่น (ปี ค.ศ.1900 ไม่เป็นปีอธิกสุรทิน แม้จะหารด้วย 4 ลงตัว แต่ปี ค.ศ. 2000 เป็นปีอธิกสุรทิน)
17. เขียนโปรแกรมเพื่อช่วยในการแปลงหน่วยการวัด โดยมีอัตราส่วนคือ 1 นิ้วเท่ากับ 2.54 เซนติเมตร ให้ผู้ใช้ระบุวิธีการแปลงค่าและเลขที่ต้องการแปลงค่า ถ้าป้อนวิธีการแปลงค่าเป็น I จะแปลงข้อมูลจากเซนติเมตรเป็นนิ้ว ถ้าป้อนวิธีการแปลงค่าเป็น C จะแปลงข้อมูลจากนิ้วเป็นเซนติเมตร หากป้อนข้อมูลวิธีการแปลงเป็นค่าอื่นหรือตัวเลขที่ต้องการแปลงค่ามีค่าติดลบ ให้ขึ้นข้อความแสดงความผิดพลาดที่เกิดขึ้น
18. อนุกรมของเลขชุดหนึ่ง คือ 1, 2, 3, 5, 8, 13, ... โดยตัวเลขถัดมาเกิดจากค่าผลบวกของ 2 เทอมที่อยู่ก่อนหน้า จงเขียนโปรแกรมโดยใช้คำสั่ง for เพื่อรับข้อมูลเลขเริ่มต้น 2 เทอมแรก และจำนวนเทอมที่ต้องการทั้งหมด และให้แสดงผลอนุกรมดังกล่าว
19. เขียนโปรแกรมเพื่อบวกเลขจำนวนเต็ม N จำนวน โดยรับค่า N จากผู้ใช้
20. เขียนโปรแกรมเพื่อบวกเลขจำนวนเต็ม N จำนวน โดยรับค่า N จากผู้ใช้
21. เขียนโปรแกรมเพื่อรับข้อมูลความสูงและอายุของผู้ใช้จำนวน 50 คน ให้นำว่ามีผู้ที่อายุมากกว่า 15 ปีและความสูงตั้งแต่ 175 เซนติเมตรขึ้นไปมีกี่คน และผู้ที่มีอายุมากกว่า 20 ปี ที่มีความสูงตั้งแต่ 180 เซนติเมตรขึ้นไปมีกี่คน และให้หาอายุเฉลี่ยและความสูงเฉลี่ยของข้อมูลที่ป้อนเข้ามาทั้งหมด
22. เขียนโปรแกรมโดยใช้คำสั่ง for เพื่อคำนวณราคาสินค้า โดยให้ผู้ใช้ป้อนจำนวนรายการของสินค้าที่ต้องการคำนวณ และป้อนรายละเอียดของแต่ละรายการประกอบด้วย ราคาต่อหน่วยและจำนวนหน่วยของสินค้า ให้คำนวณ
  - ราคาสินค้าทั้งหมด
  - ร้านค้ามีข้อกำหนดการลดราคาว่า หากซื้อตั้งแต่ 5,000 บาท ลดราคา 3 % ถ้าซื้อตั้งแต่ 40,000 บาท ลดราคา 4% และซื้อเกิน 100,000 บาท ลดราคา 5 % ให้คำนวณยอดเงินของการลดราคาและยอดเงินที่ลูกค้าต้องชำระหลังจากลดราคาแล้ว
23. เขียนโปรแกรมเพื่อตรวจสอบว่าเลขจำนวนเต็ม N ที่รับจากผู้ใช้เป็น Prime Number หรือไม่ โดย Prime Number คือ เลขจำนวนเต็มบวกที่มากกว่า 1 ซึ่งมีแต่เลข 1 และตัวมันเองเท่านั้นที่หารได้ลงตัว

24. เขียนโปรแกรมโดยใช้คำสั่ง while เพื่อคำนวณราคาสินค้า โดยให้ผู้ป้อนจำนวนรายการของสินค้าที่ต้องการคำนวณ และป้อนรายละเอียดของแต่ละรายการประกอบด้วย ราคาต่อหน่วยและจำนวนหน่วยของสินค้า ให้คำนวณ
- ราคาสินค้าทั้งหมด
  - ร้านค้ามีข้อกำหนดการลดราคาว่า ยอดเงิน 10,000 บาทแรกลดราคาให้ 2 % ยอดเงินที่เกินกว่า 10,000 บาท แต่ไม่เกิน 50,000 บาทลดราคาให้ 3 % ยอดเงินที่เกินกว่านั้นจะลดราคาให้ 4 % ให้คำนวณยอดเงินของการลดราคา และยอดเงินที่ถูกค้าต้องชำระหลังจากลดราคาแล้ว
25. ปรับปรุงโปรแกรมในข้อ 15 ใช้คำสั่ง while เช่นเดิม โดยไม่มีการป้อนจำนวนรายการของสินค้าที่ต้องการคำนวณ แต่ถ้าเมื่อใดที่ผู้ใช้ป้อนราคาต่อหน่วยของสินค้ามีค่าติดลบหรือเป็นศูนย์ แสดงว่าเป็นจุดสิ้นสุดของการป้อนรายการสินค้า
26. ปรับปรุงโปรแกรมในข้อ 12 โดยมีการทำงานวนซ้ำเพื่อรับข้อมูลใหม่ เมื่อทำงานเสร็จแต่ละรอบให้ขึ้นข้อความว่า "Continue program (Y/N) ?" หากผู้ใช้ตอบ Y ให้ทำงานแปลงข้อมูลใหม่ แต่ถ้าป้อนข้อมูลอื่น ๆ ให้จบการทำงานของโปรแกรม
27. เขียนโปรแกรมโดยใช้คำสั่ง do-while เพื่อรับข้อมูลความสูงของสมาชิกชมรมบาสเก็ตบอลที่ผู้ใช้ป้อนเข้าสู่ระบบ トラバใดที่ปริมาณความสูงเฉลี่ยสะสมยังไม่เกิน 175 เซนติเมตร และยังมีข้อมูลสมาชิกไม่เกิน 5 คน ให้รับข้อมูลไปเรื่อย ๆ ให้หาว่ามีการรับข้อมูลทั้งหมดกี่จำนวน และความสูงเฉลี่ยสะสมสุดท้ายมีค่าเท่าใด
28. เขียนโปรแกรมเพื่อรับข้อมูลคะแนนรวมของนักศึกษาในกระบวนวิชาหนึ่ง ซึ่งมีไม่เกิน 80 คน หากเมื่อใดที่ป้อนคะแนนติดลบ แสดงว่าสิ้นสุดการป้อนข้อมูล ให้หาจำนวนนักศึกษาที่ได้เกรดได้ระดับต่าง ๆ ว่ามีจำนวนกี่คน โดยมีข้อกำหนดว่า

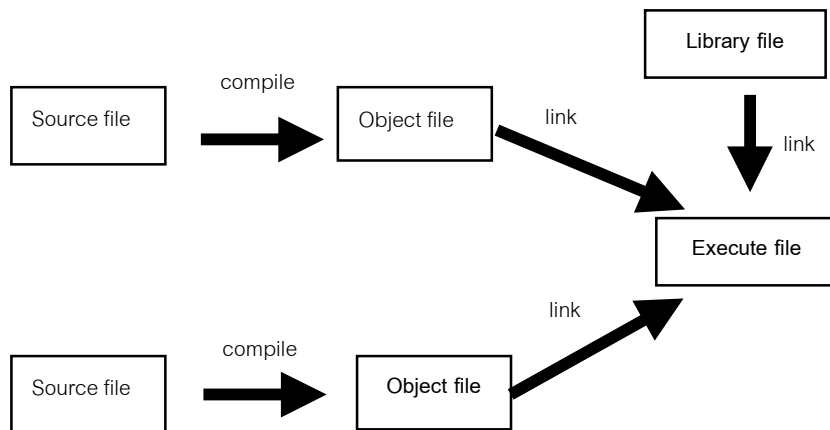
| คะแนน      | เกรด |
|------------|------|
| ต่ำกว่า 50 | F    |
| 50-59      | D    |
| 60-69      | C    |
| 70-79      | B    |
| 80 ขึ้นไป  | A    |

# ฟังก์ชัน ( Functions )

# 4

การเขียนโปรแกรมทั้งหมดภายในฟังก์ชันเดียวหากเป็นโปรแกรมขนาดใหญ่ เมื่อมีการกลับมาแก้ไข หรือต้องการปรับปรุงโปรแกรมเพิ่มเติมจะทำได้ยาก โดยเฉพาะเมื่อผู้ที่มาปรับปรุงนั้นไม่ใช่ผู้เขียนโปรแกรม การเขียนโปรแกรมแบบฟังก์ชันเป็นการกระจายการทำงานของโปรแกรมออกเป็นงานเล็ก ๆ จะสามารถช่วยแก้ปัญหาดังกล่าวได้

โปรแกรมภาษาซีจะประกอบขึ้นจากฟังก์ชันหลาย ๆ ฟังก์ชันอยู่ในแฟ้มต้นฉบับ (Source File) ตั้งแต่หนึ่งแฟ้มขึ้นไป หลังจากนั้นจะนำแฟ้มเหล่านี้ไปทำการแปลโปรแกรม (Compile) เพื่อให้ได้แฟ้มจุดหมาย (Object File) และเชื่อม (Link) เข้าด้วยกัน และอาจจะมีการเชื่อมเข้ากับคลังโปรแกรม (Library File) หากมีการเรียกใช้ฟังก์ชันอื่น ๆ จากคลังโปรแกรม คลังโปรแกรมเป็นแฟ้มจุดหมายที่เก็บฟังก์ชันย่อยที่สามารถเรียกใช้ได้โดยโปรแกรมใด ๆ ช่วยให้ประหยัดเวลาในการพัฒนาโปรแกรมได้เป็นอย่างมาก เช่น หากมีการใช้ฟังก์ชันใดบ่อยในหลาย ๆ โปรแกรม หรือหากจะต้องเขียนฟังก์ชันนั้นในทุกโปรแกรมที่เรียกใช้จะเป็นการเสียเวลาพัฒนาโปรแกรม ก็สามารถนำฟังก์ชันนั้นไปเก็บไว้ในคลังโปรแกรมเพื่อให้โปรแกรมอื่นเรียกใช้ได้



รูปที่ 4.1 แสดงขั้นตอนการแปลโปรแกรมภาษาซี

## 1. แนวความคิดในการออกแบบฟังก์ชัน

เมื่อเริ่มเขียนโปรแกรมเราจะต้องรู้ว่าโปรแกรมนั้นเขียนขึ้นเพื่อทำงานอะไร โดยที่เราสามารถหารายละเอียดเหล่านั้นจากผู้ใช้งาน จากเอกสารต่าง ๆ ที่เกี่ยวข้อง และจากแหล่งข้อมูลต่าง ๆ หลังจากนั้นจะเป็นออกแบบโปรแกรมเพื่อให้สามารถทำในสิ่งที่เราต้องการ

การออกแบบโปรแกรมในภาษาซีจะอยู่บนพื้นฐานของการออกแบบมอดูล (Module Design) โดยการแบ่งโปรแกรมออกเป็นงานย่อย ๆ (หรือมอดูล) แต่ละงานย่อยจะทำงานอย่างไรอย่างหนึ่งเท่านั้น และไม่ควรจะมีขนาดใหญ่จนเกินไป งานย่อยเหล่านี้เมื่อนำไปเขียนโปรแกรมในภาษาซีจะเป็นการเขียนในลักษณะของฟังก์ชัน

ในการออกแบบงานหรือฟังก์ชันจะอยู่บนพื้นฐานของการประมวลผลระบบคอมพิวเตอร์ คือมีข้อมูลเข้า มีกระบวนการ และมีข้อมูลออก โดยที่จะมีฟังก์ชันหลักทำหน้าที่ควบคุมและประสานการทำงาน

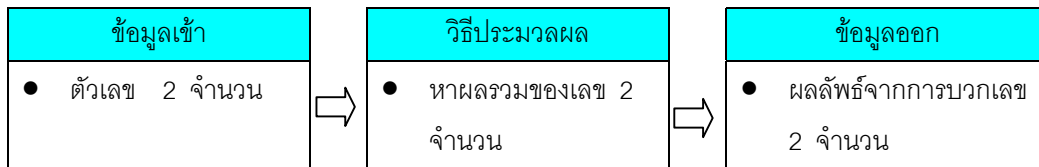
- ข้อมูลเข้า หมายถึงข้อมูลที่จำเป็นต้องทราบก่อนการประมวลผลฟังก์ชันนั้น โดยฟังก์ชันที่ออกคำสั่งจะเป็นผู้ให้ข้อมูลแก่ฟังก์ชันที่ได้รับคำสั่งให้ทำงาน
- ข้อมูลออก หมายถึงข้อมูลที่ได้จากการประมวลผลของฟังก์ชัน เมื่อฟังก์ชันที่ได้รับคำสั่งทำงานเสร็จจะส่งค่าข้อมูลนี้กลับไปให้กับฟังก์ชันที่ออกคำสั่ง

เนื่องจากภาษาซีมีการส่งอาร์กิวเมนต์ (argument) ให้กับฟังก์ชันแบบ By Value และฟังก์ชันสามารถคืนค่า (return) ค่าได้เพียงหนึ่งค่า ซึ่งสามารถอธิบายได้ดังนี้

- ในการออกคำสั่งให้ฟังก์ชันใดทำงานนั้น ฟังก์ชันที่เป็นผู้ออกคำสั่งอาจต้องมีการส่งข้อมูลบางอย่างให้กับฟังก์ชันย่อยนั้น ฟังก์ชันที่ออกคำสั่งสามารถส่งข้อมูลจำนวนเท่าใดก็ได้ให้แก่ฟังก์ชันที่ได้รับคำสั่งให้ทำงาน หรืออาจจะไม่ส่งข้อมูลใดๆ เลยก็ได้
- เมื่อฟังก์ชันที่ได้รับคำสั่งทำงานเสร็จจะสามารถส่งค่าข้อมูลกลับไปให้แก่ฟังก์ชันที่ออกคำสั่งได้เพียงหนึ่งค่าเท่านั้น หรืออาจไม่ส่งข้อมูลกลับเลยก็สามารถทำได้

**ตัวอย่างที่ 4.1** การออกแบบโปรแกรมการบวกเลขจำนวนจริง 2 จำนวนที่รับจากผู้ใช้งาน และแสดงผลลัพธ์ที่ได้จากการคำนวณ

วิเคราะห์การทำงานของโปรแกรม



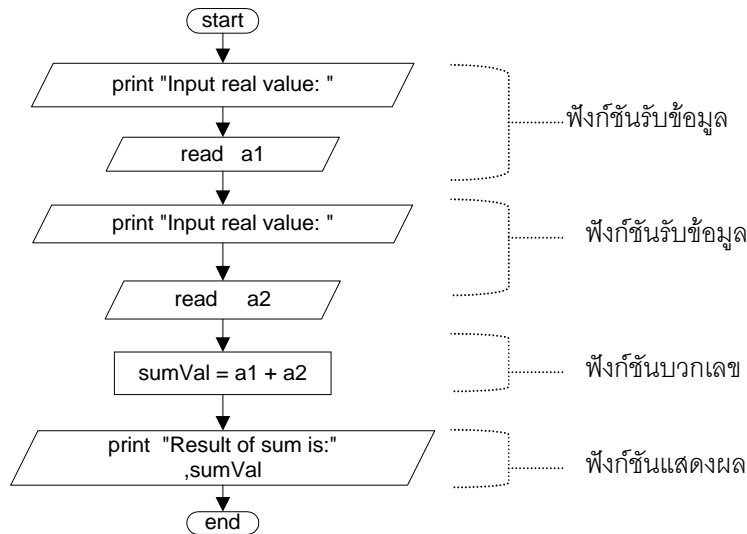
ในการออกแบบโปรแกรมนี้ สามารถแบ่งออกเป็นงานย่อย ๆ ดังนี้

- รับข้อมูล 2 จำนวนจากผู้ใช้งาน
- บวกเลข 2 จำนวนแล้วเก็บผลลัพธ์
- แสดงผลลัพธ์ของการทำงาน

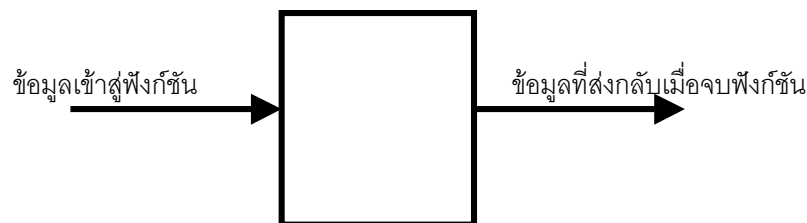
จากการแบ่งการทำงานออกเป็นงานย่อยๆ และข้อควรคำนึงที่ระบุข้างต้น เราสามารถออกแบบโปรแกรมนี้โดยแบ่งการทำงานออกเป็น 3 ฟังก์ชันย่อย และ 1 ฟังก์ชันหลัก โดยที่ 3 ฟังก์ชันย่อยดังกล่าวได้แก่

1. ฟังก์ชันรับข้อมูล ทำหน้าที่รับข้อมูลจำนวนจริงครั้งละ 1 จำนวนจากผู้ใช้งาน ( เนื่องจากแต่ละฟังก์ชันสามารถส่งข้อมูลออกได้เพียงค่าเดียว ) และส่งค่าข้อมูลที่รับเข้ามาให้กับฟังก์ชันหลัก
2. ฟังก์ชันบวกเลข ทำหน้าที่บวกเลขจำนวนจริง 2 จำนวน และส่งค่าข้อมูลที่ได้จากการคำนวณกลับมาให้ฟังก์ชันหลัก
3. ฟังก์ชันแสดงผล ทำหน้าที่แสดงผลลัพธ์จากการคำนวณ

สำหรับฟังก์ชันหลัก คือ ฟังก์ชัน main() ในภาษาซีการทำงานจะเริ่มทำงานที่ฟังก์ชัน main() เสมอ ฟังก์ชันนี้จะทำหน้าที่ควบคุมการทำงานทั้งหมด โดยฟังก์ชัน main() จะทำการเรียกฟังก์ชันย่อยต่างๆ ลำดับการเรียกใช้งานฟังก์ชันย่อยสามารถแสดงได้ดังผังงานต่อไปนี้



ในการออกแบบฟังก์ชันย่อยสามารถใช้วิธีออกแบบแบบกล่องดำ (Black Box Design) การออกแบบในลักษณะนี้จะคิดจากหน้าที่ของฟังก์ชันนั้นเป็นหลัก และพิจารณาว่าจะต้องมีข้อมูลเข้าสู่ฟังก์ชันและข้อมูลที่จะส่งกลับเมื่อจบฟังก์ชันคืออะไร ดังรูปที่ 4.2

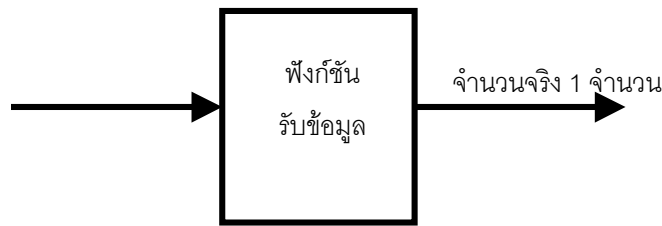


รูปที่ 4.2 การออกแบบฟังก์ชันแบบกล่องดำ

จากตัวอย่าง 4.1 เราสามารถออกแบบแต่ละฟังก์ชันย่อยทั้ง 3 ฟังก์ชันแบบกล่องดำได้ดังนี้

### ฟังก์ชันรับข้อมูล

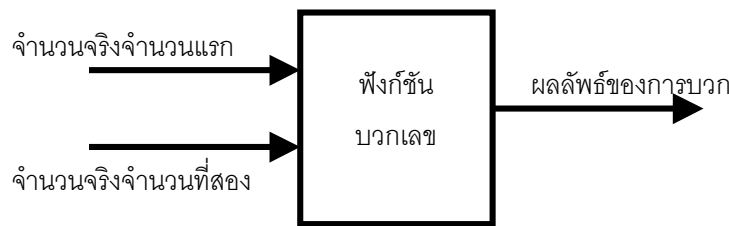
ทำหน้าที่รับข้อมูลจำนวนจริง 1 จำนวนจากผู้ใช้ เมื่อทำงานเสร็จจะทำการส่งค่าจำนวนจริงที่รับจากผู้ใช้กลับไปยังฟังก์ชันที่ออกคำสั่งให้ฟังก์ชันรับข้อมูลทำงาน ซึ่งในที่นี้คือฟังก์ชัน main() สำหรับข้อมูลเข้าสู่ฟังก์ชันในกรณีนี้จะพบว่าไม่จำเป็นต้องมีข้อมูลเข้าสู่ฟังก์ชัน การออกแบบฟังก์ชันรับข้อมูลแสดงได้ดังรูปที่ 4.3



รูปที่ 4.3 แสดงกล่องดำของฟังก์ชันรับข้อมูล

### ฟังก์ชันบวกเลข

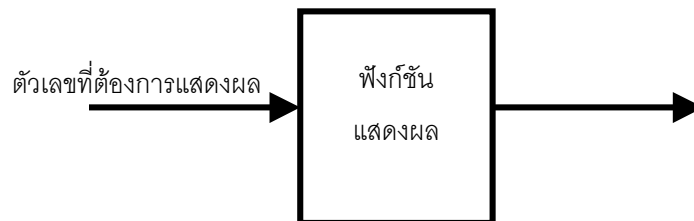
ฟังก์ชันบวกเลขมีหน้าที่ในการบวกเลข 2 จำนวน พิจารณาข้อมูลเข้าสู่ฟังก์ชันจะได้ว่าในการบวกเลขจะต้องมีการรับข้อมูลจำนวน 2 จำนวนเข้าสู่ฟังก์ชันเพื่อนำมาบวกกัน และคืนค่าผลลัพธ์ของการบวกนั้นกลับไปยังฟังก์ชันที่ออกคำสั่งให้ฟังก์ชันบวกเลขทำงาน ซึ่งในที่นี้คือฟังก์ชัน `main()` ซึ่งผลลัพธ์ของการบวกนั้นคือข้อมูลที่ออกจากฟังก์ชัน การออกแบบฟังก์ชันบวกเลขแสดงดังรูปที่ 4.4



รูปที่ 4.4 แสดงกล่องดำของฟังก์ชันบวกเลข

### ฟังก์ชันแสดงผล

ฟังก์ชันแสดงผลมีหน้าที่แสดงผลลัพธ์ออกทางจอภาพ พิจารณาข้อมูลเข้าสู่ฟังก์ชันจะได้ว่าข้อมูลที่ต้องการแสดงทางจอภาพของโปรแกรมนี้คือข้อมูลผลลัพธ์ของการบวกเลขจำนวนจริง 2 จำนวน และเมื่อแสดงผลเสร็จก็จะจบการทำงานของฟังก์ชันโดยที่ไม่จำเป็นต้องมีคืนค่าใด ๆ ไปยังฟังก์ชันที่ออกคำสั่งให้ฟังก์ชันนี้ทำงาน การออกแบบฟังก์ชันแสดงผลแสดงดังรูปที่ 4.5



รูปที่ 4.5 แสดงกล่องดำของฟังก์ชันแสดงผล

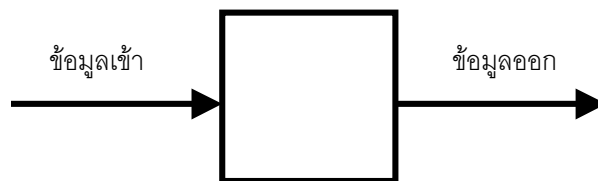
## 2. รูปแบบของฟังก์ชัน

การเขียนฟังก์ชันในภาษาซีสามารถแบ่งการเขียนออกเป็น 2 แบบหลักตามลักษณะการเขียน คือ

1. ฟังก์ชันที่มีการส่งค่ากลับเมื่อทำงานเสร็จ
2. ฟังก์ชันที่ไม่มีการส่งค่ากลับเมื่อทำงานเสร็จ

### 2.1 ฟังก์ชันที่มีการส่งค่ากลับเมื่อทำงานเสร็จ

ฟังก์ชันที่มีการส่งค่ากลับเมื่อทำงานเสร็จนี้เป็นฟังก์ชันที่จะต้องมีการส่งค่ากลับไปยังฟังก์ชันที่เรียกใช้งานฟังก์ชันนี้เมื่อทำงานเสร็จ



รูปที่ 4.6 แสดงกล่องดำของรูปแบบฟังก์ชันที่มีการส่งค่ากลับเมื่อทำงานเสร็จ

รูปแบบฟังก์ชันที่มีการส่งค่ากลับเมื่อทำงานเสร็จมีลักษณะดังนี้

```

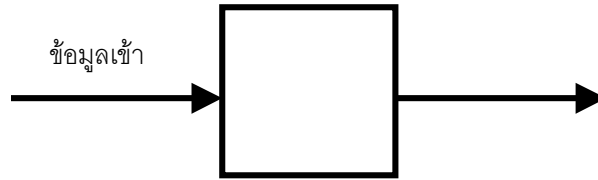
ชนิดข้อมูลที่คืนค่า ชื่อฟังก์ชัน (การประกาศตัวแปรที่จะรับค่าจากฟังก์ชันอื่นที่เรียกใช้ฟังก์ชันนี้)
{
 การประกาศตัวแปรภายในฟังก์ชัน;
 คำสั่ง;
 return(ค่าข้อมูลที่ต้องการส่งค่ากลับ);
}

```

จากรูปแบบของฟังก์ชันข้างต้น ชนิดของข้อมูลที่คืนค่า จะเป็นประเภทข้อมูลชนิดใด ๆ เช่น int float float char เป็นต้น ก่อนจบฟังก์ชันจะต้องมีการส่งค่ากลับด้วยคำสั่ง return แล้วตามด้วยค่าที่คืนกลับซึ่งเป็นข้อมูลชนิดเดียวกับ ชนิดของข้อมูลที่คืนค่า

## 2.2 ฟังก์ชันที่ไม่มีการส่งค่ากลับเมื่อทำงานเสร็จ

ฟังก์ชันที่ไม่มีการส่งค่ากลับเมื่อทำงานเสร็จ จะเป็นฟังก์ชันที่เมื่อทำงานในฟังก์ชันเสร็จก็จะคืนการทำงานกลับไปยังฟังก์ชันที่เรียกใช้งานโดยที่ไม่มีการคืนค่าใด ๆ กลับไป



รูปที่ 4.7 แสดงกล่องดำของรูปแบบฟังก์ชันที่ไม่มีการส่งค่ากลับเมื่อทำงานเสร็จ

รูปแบบฟังก์ชันที่ไม่มีการส่งค่ากลับเมื่อทำงานเสร็จมีลักษณะดังนี้

```

void ชื่อฟังก์ชัน (การประกาศตัวแปรที่จะรับค่าจากฟังก์ชันอื่นที่เรียกใช้ฟังก์ชันนี้)
{
 การประกาศตัวแปรภายในฟังก์ชัน;
 คำสั่ง;
}

```

จากรูปแบบของการประกาศฟังก์ชันที่ไม่มีการส่งค่ากลับจะเริ่มต้นด้วยคำว่า void และไม่มีคำสั่ง return ก่อนจะจบการทำงานของฟังก์ชัน

**ตัวอย่างที่ 4.2** แสดงการทำงานของโปรแกรมการบวกเลขจำนวนจริง 2 จำนวนที่รับจากผู้ใช้งาน

```

#include <stdio.h>
float InputFloat () {
 float x;
 printf ("\nInput real value : ");
 scanf ("%f", &x);
 return (x);
}
float SumFloat (float x, float y) {
 return (x + y);
}

```



```

void PrintOut (float x) {
 printf ("\n Result of sum is : %.2f", x);
}

void main () {
 float a1, a2, sumVal;
 a1 = InputFloat();
 a2 = InputFloat();
 sumVal = SumFloat (a1, a2);
 PrintOut (sumVal);
}

```

ตัวอย่างที่ 4.2 เป็นตัวอย่างที่แสดงให้เห็นถึงการประกาศฟังก์ชัน การทำงานของฟังก์ชัน การส่งผ่านค่าข้อมูล การคืนค่าจากฟังก์ชัน จะเห็นว่าฟังก์ชันแต่ละฟังก์ชันทำงานเพียงอย่างเดียวอย่างหนึ่งเท่านั้น

การทำงานเริ่มจากคำสั่งแรก `#include <stdio.h>` เป็นการเรียกใช้อินคลูซไฟล์ (Include File) เมื่อโปรแกรมมีการเรียกใช้งานฟังก์ชันจากคลังโปรแกรม ซึ่งอาจจะเป็นแฟ้มจากคลังโปรแกรมมาตรฐานที่ผู้ผลิตคอมพิวเตอร์เตรียมไว้ หรือแฟ้มจากคลังโปรแกรมที่ผู้พัฒนาโปรแกรมจัดทำขึ้นเอง จะทำการเรียกใช้แฟ้มที่เก็บฟังก์ชันที่ต้องการในคลังโปรแกรม ในที่นี้มีการเรียกใช้ฟังก์ชันที่เกี่ยวข้องกับการรับข้อมูลจากผู้ใช้และแสดงผลข้อมูลทางอุปกรณ์แสดงผล ได้แก่ `scanf` และ `printf` จึงมีการเรียกใช้แฟ้มที่เก็บฟังก์ชันที่เกี่ยวข้องกับอุปกรณ์รับและแสดงผลมาตรฐาน คือ `stdio.h` กรณีที่เป็นแฟ้มโปรแกรมมาตรฐานจะเก็บไว้ในสารบบ (directory) ที่ผู้ผลิตคอมพิวเตอร์จัดเตรียมไว้ จะใช้คำสั่ง

```
include <stdio.h>
```

แต่หากเป็นแฟ้มที่รวบรวมฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้นเองจะใช้คำสั่ง

```
include "myfile.h"
```

หากไม่มีการระบุสารบบ โปรแกรมจะค้นหาแฟ้ม `myfile.h` ที่สารบบที่ทำงานปัจจุบัน แต่ผู้เขียนโปรแกรมสามารถระบุสารบบไว้ในเครื่องหมาย “ ” ได้เช่นเดียวกัน แฟ้มที่รวบรวมฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้นเองนี้จะเป็นแฟ้มข้อความธรรมดาที่สามารถเปิดอ่านได้ เก็บข้อมูลที่จำเป็นต้องใช้ในการเรียกใช้ฟังก์ชัน เช่น การประกาศตัวแปร การประกาศโปรโตไทป์ของฟังก์ชัน เป็นต้น

จากนั้นการทำงานของโปรแกรมจะเริ่มจากฟังก์ชันหลักคือ ฟังก์ชัน `main( )` ภายในฟังก์ชัน `main( )` หากมีการเรียกใช้ฟังก์ชันย่อยใด ก็จะมีส่งการทำงานไปยังฟังก์ชันย่อยนั้น ๆ เมื่อฟังก์ชันย่อยนั้นทำงานเสร็จก็จะคืนการทำงานกลับมายังฟังก์ชัน `main( )` ทำเช่นนี้จนกระทั่งหมดคำสั่งในฟังก์ชัน `main( )` ลำดับของการเรียกใช้งานฟังก์ชันย่อยมีดังนี้

1. เรียกฟังก์ชันรับข้อมูล เพื่อรับข้อมูลตัวแรก
2. เรียกฟังก์ชันรับข้อมูล เพื่อรับข้อมูลตัวที่สอง
3. เรียกฟังก์ชันบวกเลข เพื่อทำการคำนวณ
4. เรียกฟังก์ชันแสดงผล เพื่อแสดงผลลัพธ์จากการคำนวณ

ขั้นตอนที่ 1 และ 2 เป็นขั้นตอนของการรับข้อมูลเข้าเพื่อประมวลผล การทำงานทั้งสองขั้นตอนนี้ฟังก์ชันหลักจะทำการเรียกใช้งานฟังก์ชันรับข้อมูล ซึ่งเป็นฟังก์ชันที่มีการคืนค่ากลับเป็นจำนวนจริง 1 จำนวน ฟังก์ชันรับข้อมูลมีการประกาศหัวของฟังก์ชันรูปแบบดังนี้คือ

```
float InputFloat ();
```

ในฟังก์ชันหลักหรือฟังก์ชัน main ( ) จะต้องมีกรับข้อมูลจำนวนจริง 2 ค่า เพราะฉะนั้นจะต้องมีการเรียกใช้งานฟังก์ชันนี้ 2 ครั้ง ดังขั้นตอนที่ 1 และ 2 ข้างต้น เนื่องจากฟังก์ชันนี้มีการคืนค่าเป็นจำนวนจริงแบบ float ภายในฟังก์ชัน main ( ) จะต้องมีตัวแปรแบบ float เพื่อมารับค่าดังกล่าว การเรียกใช้ฟังก์ชันนี้สามารถใช้คำสั่ง

```
float a1, a2; // ตัวแปรที่ใช้รับค่าการรับข้อมูลจากผู้ใช้
a1 = InputFloat();
a2 = InputFloat();
```

พิจารณาจากการเรียกใช้งาน หากฟังก์ชันใดมีการส่งค่ากลับไปยังฟังก์ชันที่เรียกใช้ การเรียกใช้จะต้องมีตัวแปรที่มีชนิดเดียวกับข้อมูลที่ส่งคืนจากฟังก์ชันย่อยมารับค่าเสมอ จากในตัวอย่างฟังก์ชัน InputFloat ( ) มีการคืนค่าเป็นแบบ float เพราะฉะนั้นตัวแปรที่มารับค่าจะเป็นตัวแปรประเภทเดียวกัน ในที่นี้คือ a1 เป็นตัวแปรประเภท float จะถูกกำหนดให้มีค่าเท่ากับค่าที่ส่งคืนมาจากฟังก์ชัน InputFloat ( ) ในที่นี้มีการรับค่าจำนวนจริง 2 จำนวนจากผู้ใช้ เพราะฉะนั้นจะต้องมีการเรียกใช้ฟังก์ชัน InputFloat ( ) 2 ครั้ง ครั้งแรกเก็บค่าที่ผู้ใช้ป้อนในตัวแปร a1 และเมื่อมีการเรียกใช้ครั้งที่ 2 ค่าที่ได้จะเก็บไว้ในตัวแปร a2

ขั้นตอนที่ 3 เป็นขั้นตอนของการบวกเลขจำนวนจริง 2 จำนวนที่รับเข้ามาจากผู้ใช้ การทำงานขั้นตอนนี้ฟังก์ชันหลักจะทำการเรียกใช้งานฟังก์ชันบวกเลข โดยมีการส่งค่าเลขจำนวนจริง 2 ค่าให้กับฟังก์ชันบวกเลขเพื่อทำการคำนวณ และเมื่อทำการคำนวณเสร็จฟังก์ชันบวกเลขจะส่งค่าผลลัพธ์ของการบวกกลับเป็นจำนวนจริง 1 จำนวนคืนให้กับฟังก์ชันหลัก ฟังก์ชันบวกเลขมีการประกาศหัวของฟังก์ชันรูปแบบดังนี้คือ

```
float SumFloat (float, float);
```

เมื่อต้องการเรียกใช้งานฟังก์ชันย่อยดังกล่าวจะต้องมีการส่งค่าเข้าไป 2 ค่า ในที่นี้คือ a1 และ a2 ที่รับจากผู้ใช้ มีชนิดข้อมูลเป็น float และเมื่อฟังก์ชัน SumFloat ( ) ทำงานเสร็จจะมีการส่งค่ากลับจึงต้องมีตัวแปรชนิด float อีก 1 ตัวมารับค่าผลบวกดังกล่าว จะเรียกใช้งานฟังก์ชัน SumFloat ได้ว่า

```
float sumVal;
sumVal = SumFloat (a1, a2);
```

ขั้นตอนที่ 4 เป็นขั้นตอนของแสดงผลลัพธ์ของการคำนวณ การทำงานขั้นตอนนี้ฟังก์ชันหลักจะทำการเรียกใช้งานฟังก์ชันแสดงผล โดยมีการส่งค่าเลขจำนวนจริง 1 ค่าซึ่งเป็นผลลัพธ์ให้กับฟังก์ชันนี้เพื่อพิมพ์ผลลัพธ์ ซึ่งการทำงานของฟังก์ชันนี้ไม่จำเป็นต้องมีการส่งค่ากลับให้ฟังก์ชันหลัก ฟังก์ชันแสดงผลมีการประกาศหัวของฟังก์ชันรูปแบบดังนี้คือ

```
void PrintOut (float);
```

ฟังก์ชันนี้มีการรับค่าจำนวนจริง 1 ค่าเป็นแบบ float โดยที่เมื่อทำงานเสร็จจะไม่มีค่าส่งกลับ เมื่อต้องการเรียกใช้งาน สามารถเรียกใช้ได้ด้วยคำสั่ง

```
PrintOut(sumVal);
```

จะเห็นว่าการเรียกใช้งานฟังก์ชันที่ไม่มีค่าส่งกลับนั้นสามารถเรียกใช้ชื่อฟังก์ชันได้ทันทีโดยไม่ต้องมีตัวแปรมารับค่า

### 3. การประกาศโปรโตไทป์ของฟังก์ชัน

การประกาศโปรโตไทป์เป็นสิ่งจำเป็นในภาษาซี เนื่องจากภาษาซีเป็นภาษาในลักษณะที่ต้องมีการประกาศฟังก์ชันก่อนจะเรียกใช้ฟังก์ชันนั้น (Predefined Function) จากตัวอย่างที่ 4.2 จะเห็นว่าฟังก์ชัน main ( ) จะอยู่ใต้ฟังก์ชันอื่น ๆ ที่มีการเรียกใช้ นั่นคือมีการประกาศฟังก์ชันย่อยไว้ในบรรทัดก่อนที่จะมีการเรียกใช้ แต่หากต้องการย้ายฟังก์ชัน main ( ) ขึ้นไปไว้ด้านบน จะต้องมีการประกาศโปรโตไทป์ของฟังก์ชันที่ต้องการเรียกใช้ก่อนเสมอ ดังตัวอย่างที่ 4.3 ซึ่งผู้เขียนโปรแกรมจะเลือกวิธีการเขียนในรูปแบบในตัวอย่างที่ 4.2 หรือ 4.3 ก็ได้

**ตัวอย่างที่ 4.3** แสดงการทำงานของโปรแกรมการบวกเลขจำนวนจริง 2 จำนวนที่รับจากผู้ใช้ในลักษณะที่มีการประกาศโปรโตไทป์

```
#include <stdio.h>

float InputFloat ();
float SumFloat (float , float);
void PrintOut (float);

void main () {
 float a1, a2, sumVal;
 a1 = InputFloat();
 a2 = InputFloat();
 sumVal = SumFloat (a1, a2);
 PrintOut (sumVal);
}

float InputFloat () {
 float x;
 printf ("\nInput real value : ");
 scanf ("%f", &x);
 return (x);
}

float SumFloat (float x, float y) {
 return (x + y);
}
```

```
void PrintOut (float x) {
 printf ("\n Result of sum is : %.2f", x);
}
```

---

จะเห็นว่าในโปรโตไทป์ไม่มีการประกาศชื่อตัวแปร มีแต่การเขียนประเภทของตัวแปรไว้ภายใน เป็นการช่วยให้คอมไพเลอร์สามารถตรวจสอบจำนวนของตัวแปร ประเภทของตัวแปร ประเภทของการคืนค่า ภายในโปรแกรมว่ามีการเรียกใช้งานสิ่งต่าง ๆ เกี่ยวกับฟังก์ชันนั้นถูกต้องหรือไม่ นอกจากนี้เราอาจจะแยกส่วนโปรโตไทป์ไปเขียนไว้ในอินคลูซไฟล์ก็ได้เช่นเดียวกัน ตัวอย่างที่ 4.4

**ตัวอย่างที่ 4.4** แสดงการแยกส่วนของโปรโตไทป์ไปไว้ในอินคลูซไฟล์ชื่อ myinc.h และการเรียกใช้จากโปรแกรมหลัก

---

```
/* myinc.h */
float InputFloat ();
float SumFloat (float , float);
void PrintOut (float);
```

---

```
#include <stdio.h>
#include "myinc.h"

void main () {
 float a1, a2, sumVal;
 a1 = InputFloat();
 a2 = InputFloat();
 sumVal = SumFloat (a1, a2);
 PrintOut (sumVal);
}

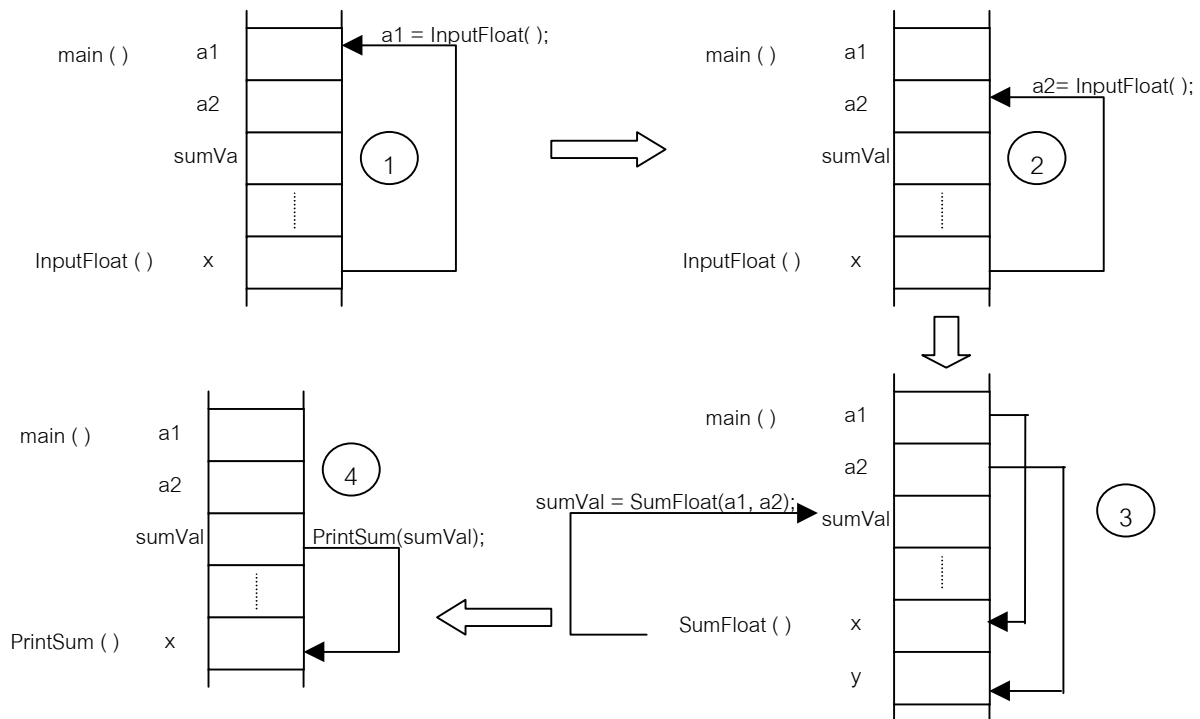
float InputFloat () {
 float x;
 printf ("\nInput real value : ");
 scanf ("%f", &x);
 return (x);
}

float SumFloat (float x, float y) {
 return (x + y);
}
```

```
void PrintOut (float x) {
 printf ("\n Result of sum is : %.2f", x);
}
```

#### 4. ขอบเขต (Scope)

การทำงานของโปรแกรมภาษาซีจะทำงานที่ฟังก์ชัน main ( ) ก่อนเสมอ เมื่อฟังก์ชัน main ( ) เรียกใช้งานฟังก์ชันอื่น ก็จะมีการส่งคอนโทรล (Control) ที่ควบคุมการทำงานไปยังฟังก์ชันนั้น ๆ จนกว่าจะจบฟังก์ชัน หรือพบคำสั่ง return เมื่อมีการเรียกใช้งานฟังก์ชันจะมีการจองพื้นที่หน่วยความจำสำหรับตัวแปรที่ต้องใช้ภายในฟังก์ชันนั้น และเมื่อสิ้นสุดการทำงานของฟังก์ชันก็จะมีการคืนพื้นที่หน่วยความจำส่วนนั้นกลับสู่ระบบ การใช้งานตัวแปรแต่ละตัวจะมีขอบเขตของการใช้งานขึ้นอยู่กับตำแหน่งที่ประกาศตัวแปรนั้น จากตัวอย่างที่ 4.2 และ 4.3 แสดงการทำงานได้ดังรูปที่ 4.8



รูปที่ 4.8 แสดงการทำงานของฟังก์ชัน

เมื่อเริ่มการทำงานที่ฟังก์ชัน main ( ) จะมีการจองพื้นที่หน่วยความจำให้กับตัวแปรต่าง ๆ ที่เกี่ยวข้อง คือ a1 a2 และ sumVal เมื่อมีการเรียกใช้ฟังก์ชัน InputFloat ( ) ในขั้นที่ 1 จะมีการจองพื้นที่หน่วยความจำให้กับตัวแปร x และทำงานรับข้อมูลจากผู้ใช้ เมื่อมีการใช้คำสั่ง return ( x ); จะมีการนำค่าในตัวแปร x กำหนดให้กับ a1 แล้วคืนพื้นที่เพื่อการทำงานของ InputFloat ให้กับระบบ

ในขั้นตอนที่ 2 เมื่อมีการเรียกใช้ InputFloat อีกครั้งหนึ่ง จะมีการจองพื้นที่ในหน่วยความจำขึ้นใหม่ โดยมีตัวแปร x เช่นเดียวกัน แต่เป็นตัวแปรคนละตัวกับขั้นตอนที่ 1 และมีการทำงานเหมือนขั้นตอนที่ 1

เมื่อมีการสั่งให้คำนวณค่าในขั้นตอนที่ 3 ด้วยฟังก์ชัน SumFloat ( ) จะมีการจองพื้นที่ให้กับตัวแปร x และ y และกำหนดค่า a1 ให้กับ x และกำหนด a2 ให้กับ y ฟังก์ชันนี้จะบวกค่าใน x กับ y แล้วคืนค่าผลลัพธ์การคำนวณให้กับตัวแปร sumVal

ในขั้นตอนสุดท้ายเป็นการแสดงผลลัพธ์ด้วยฟังก์ชัน PrintSum ( ) จะมีการจองพื้นที่ให้กับตัวแปร x เช่นเดียวกัน และกำหนดค่าของ sumVal ให้กับตัวแปร x และแสดงค่านั้น

จะเห็นว่าตัวแปร x ที่ประกาศในแต่ละขั้นตอนจะทำงานอยู่ภายในฟังก์ชันที่มีการประกาศค่าเท่านั้น และใช้พื้นที่ในการเก็บข้อมูลคนละส่วนกัน ขอบเขตการทำงานของตัวแปรแต่ละตัวจะกำหนดอยู่ภายบล็อกของคำสั่งภายในเครื่องหมายปีกกา ( { } ) หรือการประกาศในช่วงของการประกาศฟังก์ชัน เรียกตัวแปรเหล่านี้ว่า ตัวแปรเฉพาะที่ (Local Variable) นอกจากนี้สามารถประกาศตัวแปรไว้ที่ภายนอกฟังก์ชันบริเวณส่วนเริ่มของโปรแกรมจะเรียกว่า ตัวแปรส่วนกลาง (Global Variable) ซึ่งเป็นตัวแปรที่สามารถเรียกใช้ที่ตำแหน่งใด ๆ ในโปรแกรมก็ได้ ยกเว้นในกรณีที่มีการประกาศตัวแปรที่มีชื่อเดียวกันตัวแปรส่วนกลางภายในบล็อกหรือฟังก์ชัน แสดงดังตัวอย่างที่ 4.5

**ตัวอย่างที่ 4.5** แสดงการทำงานของโปรแกรมในลักษณะที่มีตัวแปรส่วนกลาง แสดงขอบเขตการใช้งานของตัวแปรภายในโปรแกรม

---

```
#include <stdio.h>
int x;
void func1 () {
 x = x + 10;
 printf ("func1 -> x : %d\n", x); /* x is 20 */
}
void func2 (int x) {
 x = x + 10;
 printf ("func2 -> x : %d\n", x); /* x is 30 */
}
void func3 () {
 int x=0;
 x = x + 10;
 printf ("func3 -> x : %d\n", x); /* x is 10 */
}

void main () {
 x = 10;
 printf ("main (start) -> x : %d\n", x); /* x is 10 */
 func1 ();
 printf ("main (after func1) -> x : %d\n", x); /* x is 20 */
 func2 (x);
 printf ("main (after func2) -> x : %d\n", x); /* x is 20 */
 func3 ();
 printf ("main (after func3) -> x : %d\n", x); /* x is 20 */
}

```

---

### ผลการทำงานของโปรแกรม

```
main (start) -> x : 10
func1 -> x : 20
main (after func1) -> x : 20
func2 -> x : 30
main (after func2) -> x : 20
func3 -> x : 10
main (after func3) -> x : 20
```

การทำงานของตัวแปรส่วนกลาง x เริ่มต้นมีการกำหนดค่าในฟังก์ชัน main ( ) ให้ตัวแปรส่วนกลาง x มีค่า 10 เมื่อมีการเรียกฟังก์ชัน func1 ( ) ตัวแปร x ภายในฟังก์ชันนี้เป็นตัวแปรตัวเดียวกับตัวแปรส่วนกลาง x เมื่อมีการเพิ่มค่าขึ้น 10 ค่าของตัวแปรส่วนกลางจึงเปลี่ยนเป็น 20

เมื่อมีการเรียกใช้ฟังก์ชัน func2 ( ) โดยส่งตัวแปรส่วนกลาง x เป็นอาร์กิวเมนต์ให้กับฟังก์ชัน func2 ( ) จะมีการจองพื้นที่ให้กับตัวแปร x ซึ่งเป็นตัวแปรเฉพาะที่ใช้ได้เฉพาะภายในฟังก์ชัน func2 ( ) เท่านั้น โดยมีการกำหนดให้ตัวแปรเฉพาะที่ x มีค่าเท่ากับตัวแปรส่วนกลาง x เพราะฉะนั้นการเปลี่ยนแปลงค่าที่ตัวแปรเฉพาะที่ x ในฟังก์ชัน func2 ( ) จึงไม่มีผลกับตัวแปรส่วนกลาง x

ต่อมามีการเรียกใช้ฟังก์ชัน func3 ( ) ภายในฟังก์ชันนี้มีการประกาศตัวแปร x ตัวแปรนี้เป็นตัวแปรเฉพาะที่ใช้ได้เฉพาะภายในฟังก์ชัน func3 ( ) เพราะฉะนั้นการอ้างถึงตัวแปร x ภายในฟังก์ชันนี้จึงไม่มีส่วนเกี่ยวข้องกับตัวแปรส่วนกลาง x

การประกาศตัวแปรส่วนกลางเป็นสิ่งที่ควรระวังเนื่องจากหากโปรแกรมมีขนาดใหญ่ จะทำให้การบำรุงรักษาโปรแกรมทำได้ยาก การแก้ไขโปรแกรมที่ตำแหน่งหนึ่งอาจจะกระทบกับตัวแปรส่วนกลางที่ตำแหน่งอื่นได้ เนื่องจากการตรวจสอบว่ามีการใช้ตัวแปรส่วนกลางที่ใดบ้างในโปรแกรมจะทำได้ยากมาก

### 5. ฟังก์ชันแบบเรียกซ้ำ (Recursive Functions)

ฟังก์ชันแบบเรียกตัวเอง เป็นฟังก์ชันที่สามารถกำหนดรูปแบบของฟังก์ชันในเทอมใด ๆ กับค่าของเทอมในก่อนหน้าได้ในรูปแบบที่แน่นอน โดยที่จะต้องมีจุดสิ้นสุดของฟังก์ชันที่แน่นอน ตัวอย่างของฟังก์ชันที่สามารถทำงานในรูปของฟังก์ชันแบบเรียกตัวเองได้ เช่น ฟังก์ชันแฟคทอเรียล (Factorial)

$$\begin{aligned} 0! &= 1 \\ 1! &= 1 \\ 2! &= 2 * 1 \\ 3! &= 3 * 2 * 1 \\ &\dots \end{aligned}$$

ซึ่งสามารถเขียนฟังก์ชันแฟคทอเรียลในรูปแบบของความสัมพันธ์ได้ดังนี้

$$\begin{aligned} n! &= 1 && \text{ในกรณีที่ } n = 0 \\ n! &= n * (n-1) * (n-2) * \dots * 3 * 2 * 1 && \text{ในกรณีที่ } n > 0 \end{aligned}$$

กรณีที่  $n = 0$  เป็นจุดสิ้นสุดของการเรียกตัวเอง ในกรณีของแฟคทอเรียลฟังก์ชัน แสดงโปรแกรมดังตัวอย่างที่ 4.6

#### ตัวอย่างที่ 4.6 แสดงแฟคทอเรียลฟังก์ชัน ในลักษณะของฟังก์ชันเรียกตัวเอง

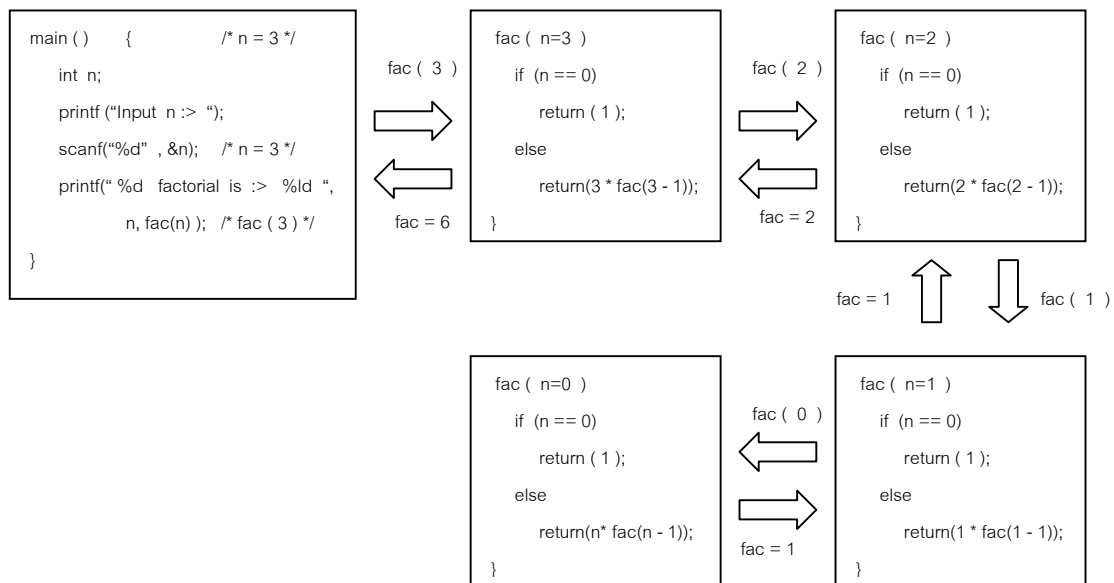
```
#include <stdio.h>

long int fac (int n) {
 if (n == 0)
 return (1);
 else
 return (n * fac(n-1));
}

main () {
 int n;
 printf ("Input n :> ");
 scanf ("%d" , &n);
 printf (" %d factorial is :> %ld ", n, fac(n));
}

```

สมมติให้  $n = 3$  สามารถแสดงการทำงานได้ดังรูปที่ 4.9



รูปที่ 4.9 แสดงการทำงานของแฟคทอเรียลฟังก์ชัน

การเขียนฟังก์ชันเรียกตัวเองจะเหมาะกับงานที่มีลักษณะเฉพาะเท่านั้น การเขียนฟังก์ชันในลักษณะเรียกตัวเองช่วยให้อ่านโปรแกรมได้เข้าใจมากขึ้น แต่จะทำให้เปลืองทรัพยากรในการทำงานเพิ่มขึ้นและการทำงานช้ากว่าโปรแกรมที่เขียนในลักษณะปกติ แสดงดังตัวอย่างที่ 4.7



#### ตัวอย่างที่ 4.7 แสดงแฟคทอเรียลฟังก์ชัน ในลักษณะของฟังก์ชันปกติ

```
#include <stdio.h>
long int fac (int n) {
 long int result=1;
 for (int i = n; i > 0; i--)
 result *= i;
 return (result);
}
main () {
 int n;
 printf ("Input n :> ");
 scanf ("%d" , &n);
 printf (" %d factorial is :> %ld ", n, fac(n));
}
```

ตัวอย่างเพิ่มเติมในเรื่องฟังก์ชันแบบเรียกซ้ำแสดงดังตัวอย่างที่ 4.8

#### ตัวอย่างที่ 4.8 การหาค่าเลขยกกำลังด้วยฟังก์ชันแบบเรียกซ้ำ

```
#include <stdio.h>
float power (float val, int pow) {
 if (pow == 0)
 return(1.0);
 else
 return(power(val, pow-1) * val);
}
void main() {
 float b=10.0, ans;
 int p=4;
 ans = power (b, p);
 printf ("Answer of %f power %d is %f", b, p, ans);
}
```

## 6. ตัวอย่างเพิ่มเติม

ในหัวข้อนี้เป็นตัวอย่างเพิ่มเติมในเรื่องของฟังก์ชัน ซึ่งจะมีการแยกฟังก์ชันตามที่โจทย์ระบุเพื่อเป็นแนวทางในการเขียนโปรแกรม ทั้งนี้หากต้องการแยกฟังก์ชันเพิ่มมากกว่าในตัวอย่างก็สามารถทำได้ตามความต้องการและความเหมาะสมเพื่อเสริมสร้างประสบการณ์ในการเขียนโปรแกรม ตัวอย่างเพิ่มเติมแสดงดังตัวอย่างที่ 4.9 ถึง 4.11

**ตัวอย่างที่ 4.9** ให้เขียนโปรแกรมเพื่อรับข้อมูลจำนวนชั่วโมง นาที และวินาทีจากผู้ใช้ และให้คำนวณว่าเวลาทั้งหมดที่รับเข้ามานั้นเป็นกี่วินาที โดยให้เขียนส่วนของกรคำนวณวินาทีเป็นฟังก์ชัน

---

```
#include <stdio.h>
#include <conio.h>

long convert(long h, long m, long s) {
 long total;
 total = (h * 60 + m) * 60 + s;
 return (total);
}

void main() {
 long hours, minutes, seconds;
 long time;

 clrscr();
 printf("Enter the time to be converted (hh mm ss) : ");
 scanf("%ld %ld %ld", &hours, &minutes, &seconds);

 time = convert(hours, minutes, seconds);
 printf("\nConvert to %ld seconds", time);
 getch();
}
```

---

**ตัวอย่างที่ 4.10** เขียนโปรแกรมเพื่อแสดงรูปกรอบสี่เหลี่ยมโดยใช้เครื่องหมาย \* ให้ผู้ใช้ระบุความกว้างและความสูงของรูปสี่เหลี่ยมที่ต้องการ

---

```
#include <stdio.h>
#include <conio.h>
void printHead(int width) {
 int i;
 for (i=0; i < width+2; i++)
 printf("***");
 printf("\n");
}
void printBody(int width, int height) {
 int i, j;
 for (i=0; i < height; i++) {
 printf("***");
 for (j=0; j < width; j++)
 printf(" ");
 printf("\n");
 }
}
void main() {
 int w, h;
 clrscr();
 printf("Enter width : ");
 scanf("%d", &w);
 printf("Enter height : ");
 scanf("%d", &h);
 printf("\n\n");
 printHead(w);
 printBody(w, h);
 printHead(w);
 getch();
}
```

---

**ตัวอย่างที่ 4.11** ให้เขียนโปรแกรมเพื่อรับข้อมูลจำนวนเต็ม 2 จำนวนจากผู้ใช้ โดยที่ข้อมูลนั้นต้องมากกว่า 20 และน้อยกว่า 60 และหาผลคูณของข้อมูลทั้ง 2

---

```
#include <stdio.h>
#include <conio.h>
#define YES 1
#define NO 0
int checkNumber(int, int);
int multiply(int, int);
void main() {
 int x, y, ans;
 int correct=NO;
 clrscr();
 do {
 printf("Enter x : ");
 scanf("%d", &x);
 printf("Enter y : ");
 scanf("%d", &y);
 correct = checkNumber(x, y);
 } while (!correct);
 ans = multiply(x, y);

 printf("\nMultiply of %d and %d is %d", x, y, ans);
 getch();
}
int checkNumber(int a, int b) {
 if (a > 20 && a < 60 && b > 20 && b < 60)
 return (YES);
 else {
 printf("Input number in range > 20 and <60\n");
 return(NO);
 }
}
int multiply(int num1, int num2) {
 return(num1 * num2);
}
```

---

## 7. ไบเบรารีมาตรฐานของภาษาซี

นอกเหนือจากฟังก์ชันต่าง ๆ ที่ผู้ใช้สามารถเขียนขึ้น ภาษาซีได้เตรียมฟังก์ชันมาตรฐานซึ่งจัดเก็บอยู่ในลักษณะที่เรียกว่าคลังโปรแกรมหรือไลบรารี (Library) ตัวอย่างของฟังก์ชันในไลบรารีมาตรฐานที่ได้มีการยกตัวอย่างไปแล้วคือ scanf( ) ที่ใช้ในการรับข้อมูล และ printf( ) ที่ใช้ในการแสดงผลข้อมูล และยังประกอบด้วยฟังก์ชันอื่น ๆ ที่คอมพิวเตอร์ในภาษาซีของทุกบริษัทผู้ผลิตจะต้องมี และคอมพิวเตอร์ของบางบริษัทอาจจะมีการจัดฟังก์ชันให้อยู่ในกลุ่มของอินคลูซไฟล์ที่ต่างกัน ซึ่งสามารถตรวจสอบได้รายละเอียดจากคู่มือของคอมพิวเตอร์แต่ละบริษัท ในที่นี้จะยกตัวอย่างเฉพาะฟังก์ชันในไลบรารีมาตรฐานที่น่าสนใจดังนี้

### 7.1 ฟังก์ชันเกี่ยวกับตัวอักษร

เมื่อต้องการใช้ฟังก์ชันเกี่ยวกับตัวอักษร จะต้องมีการเรียกใช้อินคลูซไฟล์ <ctype.h> ด้วยคำสั่ง

```
#include <ctype.h>
```

| ฟังก์ชัน       | หน้าที่                                                                                                                |
|----------------|------------------------------------------------------------------------------------------------------------------------|
| int isalnum(c) | ตรวจสอบตัวอักษรนั้นเป็นตัวอักษร 0-9 หรือ a-z หรือ A-Z หรือไม่ (Alphanumeric) ถ้าไม่ใช่คืนค่า 0 ถ้าใช่คืนค่าที่ไม่ใช่ 0 |
| int isalpha(c) | ตรวจสอบตัวอักษรนั้นเป็นตัวอักษร a-z หรือ A-Z หรือไม่ ถ้าไม่ใช่คืนค่า 0 ถ้าใช่คืนค่าที่ไม่ใช่ 0                         |
| int isdigit(c) | ตรวจสอบตัวอักษรนั้นเป็นตัวอักษร 0-9 หรือไม่ ถ้าไม่ใช่คืนค่า 0 ถ้าใช่คืนค่าที่ไม่ใช่ 0                                  |
| int islower(c) | ตรวจสอบตัวอักษรนั้นเป็นตัวอักษร a-z หรือไม่ ถ้าไม่ใช่คืนค่า 0 ถ้าใช่คืนค่าที่ไม่ใช่ 0                                  |
| int isupper(c) | ตรวจสอบตัวอักษรนั้นเป็นตัวอักษร A-Z หรือไม่ ถ้าไม่ใช่คืนค่า 0 ถ้าใช่คืนค่าที่ไม่ใช่ 0                                  |
| int tolower(c) | แปลงค่าอักขระปัจจุบันให้เป็นอักขระเดียวกันแต่เป็นตัวพิมพ์เล็ก                                                          |
| int toupper(c) | แปลงค่าอักขระปัจจุบันให้เป็นอักขระเดียวกันแต่เป็นตัวพิมพ์ใหญ่                                                          |

### 7.2 ฟังก์ชันอรรถประโยชน์ทั่วไป

การเรียกใช้งานฟังก์ชันอรรถประโยชน์ทั่วไปจะต้องเรียกใช้อินคลูซไฟล์ <stdlib.h>

| ฟังก์ชัน                  | หน้าที่                                                                                                                                                                                                                                                                       |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int abs(int n)            | หาค่าสัมบูรณ์ของค่า n                                                                                                                                                                                                                                                         |
| float atof(const char *s) | แปลงค่าสตริง s เป็นค่าข้อมูลชนิด float                                                                                                                                                                                                                                        |
| int atoi(const char *s)   | แปลงค่าสตริง s เป็นค่าข้อมูลชนิด int                                                                                                                                                                                                                                          |
| long atol(const char *s)  | แปลงค่าสตริง s เป็นค่าข้อมูลชนิด long                                                                                                                                                                                                                                         |
| void exit(int status)     | เป็นคำสั่งเพื่อใช้ในการสิ้นสุดการทำงานของโปรแกรม โดยมีการคืนค่าสถานะของการทำงานของโปรแกรมให้กับระบบ โดยทั่วไปหากโปรแกรมทำงานสำเร็จไม่มีข้อผิดพลาดจะคืนค่า 0 แต่หากมีข้อผิดพลาดจะคืนค่าที่ไม่ใช่ 0 ซึ่งผู้เขียนโปรแกรมอาจจะกำหนดค่าต่าง ๆ ขึ้นให้สอดคล้องกับความผิดพลาดที่เกิด |

| ฟังก์ชัน                              | หน้าที่                                                                                                                                                                                                                                |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| void *malloc(size_t size)             | ใช้จองพื้นที่ในหน่วยความจำมีขนาดเท่ากับ size ไบต์ คืนค่าเป็นพอยน์เตอร์ชี้ไปยังพื้นที่ถ้าจองได้ แต่ถ้าไม่สามารถจองพื้นที่ได้จะคืนค่า NULL                                                                                               |
| int rand(void)                        | คืนค่าเลขจำนวนเต็มที่สูงขึ้นมาอยู่ในช่วง 0 ถึง RAND_MAX ซึ่งไม่เกิน 32767                                                                                                                                                              |
| void *realloc(void *ptr, size_t size) | ใช้จัดการพื้นที่ในหน่วยความจำที่มีการจองไว้ก่อนหน้านี้ด้วยคำสั่ง malloc( ) ในตัวแปร ptr ใหม่ ด้วยขนาด size ไบต์ ถ้าไม่สามารถจองพื้นที่ใหม่ได้จะคืนค่า NULL ปกติจะเพื่อการเพิ่มหรือลดขนาดของพื้นที่ในหน่วยความจำที่ได้จองไว้ก่อนหน้านี้ |

### 7.3 ฟังก์ชันเกี่ยวข้องกับการรับและแสดงผลข้อมูล

การใช้งานฟังก์ชันเกี่ยวข้องกับการรับและแสดงผลข้อมูล จะต้องมีการเรียกใช้อินคลูซไฟล์ <stdio.h>

| ฟังก์ชัน                                           | หน้าที่                                                                                                                  |
|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| int getchar( )                                     | อ่านข้อมูล 1 อักขระจากอุปกรณ์รับข้อมูลมาตรฐาน                                                                            |
| char *gets(const char *buffer)                     | อ่านข้อมูลสตริงจากอุปกรณ์รับข้อมูลมาตรฐานมาเก็บในตัวแปร buffer จนกว่าจะมีการขึ้นบรรทัดใหม่ หากมีความผิดพลาดจะคืนค่า NULL |
| int printf(const char *format, ...)                | ใช้แสดงผลข้อมูลทางอุปกรณ์แสดงผลมาตรฐานในรูปแบบที่กำหนด                                                                   |
| int putchar(int c)                                 | ส่งอักขระ c ไปแสดงผลทางอุปกรณ์แสดงผลมาตรฐาน                                                                              |
| int puts(const char *buffer)                       | ส่งข้อมูลสตริง buffer ไปแสดงผลทางอุปกรณ์แสดงผลมาตรฐาน                                                                    |
| int scanf(const char *format,...)                  | ใช้อ่านข้อมูลทางอุปกรณ์รับข้อมูลมาตรฐานในรูปแบบที่กำหนด                                                                  |
| int sprintf(char *buffer, const char *format, ...) | ทำหน้าที่คล้ายกับ printf( ) แต่จะส่งผลของข้อมูลในรูปแบบที่กำหนดไปเก็บไว้ในสตริง buffer                                   |

### 7.4 ฟังก์ชันคณิตศาสตร์

การใช้งานฟังก์ชันคณิตศาสตร์ จะต้องเรียกใช้อินคลูซไฟล์ <math.h>

| ฟังก์ชัน    | หน้าที่                                    |
|-------------|--------------------------------------------|
| acos(x)     | หาค่า Arc Cosine ของ x                     |
| asin(x)     | หาค่า Arc Sine ของ x                       |
| atan(x)     | หาค่า Arc Tangent ของ x                    |
| atan2(x, y) | หาค่า Arc Tangent ของ x / y                |
| ceil(x)     | หาค่าจำนวนจริง x ที่มีการปัดเศษขึ้นทั้งหมด |
| cos(x)      | หาค่า Cosine ของ x                         |
| exp(x)      | หาค่าเอ็กโปเนนเชียล (Exponential) ของ x    |
| floor(x)    | หาค่าจำนวนจริง x ที่มีการปัดเศษขึ้นทั้งหมด |
| log(x)      | หาค่า Logarithm ของ x                      |
| log10(x)    | หาค่า Logarithm ฐาน 10 ของ x               |

| ฟังก์ชัน  | หน้าที่                 |
|-----------|-------------------------|
| pow(x, y) | หาค่า x ยกกำลัง y       |
| sin(x)    | หาค่า Sine ของ x        |
| sqrt(x)   | หาค่า Square Root ของ x |
| tan(x)    | หาค่า Tangent ของ x     |

**ตัวอย่างที่ 4.12** ให้รับข้อมูลเลขจำนวนจริงจากผู้ใช้ และแปลงค่าเป็นเลขจำนวนเต็ม โดยให้มีการปัดเศษ หากเศษมีค่ามากกว่า .50 ให้ปัดเศษขึ้น แต่ถ้าน้อยกว่าให้ปัดเศษลง

---

```
#include <stdio.h>
#include <conio.h>

void main() {
 float f;
 char s[21];
 int i;
 clrscr();
 printf("Enter number : ");
 scanf("%f", &f);
 sprintf(s, "%.0f", f);
 i = atoi(s);
 printf("Integer number is %d", i);
 getch();
}
```

---





## 2. หาผลลัพธ์ของการทำงานของโปรแกรมต่อไปนี้

(ก) #include &lt;stdio.h&gt;

```

int a=20;
void func1(int a) {
 int x;
 x = a*2;
 printf("\n2. x = %d, a = %d", x, a);
}
int func2() {
 int x;
 printf("\n4. x = %d, a = %d", x, a);
 x = a*5;
 printf("\n5. x = %d, a = %d", x, a);
 return(x);
}
void main() {
 int x=5;
 printf("\n1. x = %d, a = %d", x, a);
 func1(x);
 printf("\n3. x = %d, a = %d", x, a);
 x = func2();
 printf("\n6. x = %d, a = %d", x, a);
}

```

(ข) #include &lt;stdio.h&gt;

```

int a=10;
int func2(int x) {
 x = x - a;
 printf("\n3. x = %d, a = %d", x, a);
 return(x);
}
int func1(int a) {
 int x;
 x = a + 5;
 printf("\n2. x = %d, a = %d", x, a);
 a = func2(x*2);
 printf("\n4. x = %d, a = %d", x, a);
 return(a);
}
void main() {
 int x=0, y=5;
 printf("\n1. x = %d, y = %d, a = %d", x, y, a);
 x = func1(x);
 printf("\n5. x = %d, y = %d, a = %d", x, y, a);
 y = func2(a);
 printf("\n6. x = %d, y = %d, a = %d", x, y, a);
 y = func1(a);
 printf("\n7. x = %d, y = %d, a = %d", x, y, a);
}

```

- เขียนโปรแกรมเพื่อทำหน้าที่หาผลลัพธ์ของการคำนวณจากสมการ  $x = 2ab + c$  โดยเขียนฟังก์ชันเพื่อรับข้อมูลเลขจำนวนเต็ม a b และ c ฟังก์ชันเพื่อคำนวณผลลัพธ์จากสมการ และฟังก์ชันเพื่อแสดงผลลัพธ์
- เขียนโปรแกรมเพื่อทำหน้าที่รับข้อมูลเลขจำนวนจริง และคำนวณว่าเลขนั้นอยู่ในช่วง 100 ถึง 200 หรือไม่ ถ้าเลขดังกล่าวอยู่นอกช่วงที่ระบุให้วนรับข้อมูลนั้นใหม่ จนกว่าจะได้เลขในช่วงที่ต้องการ ให้ตรวจสอบว่ามีกรบ้อนข้อมูลทั้งหมดกี่ครั้งจึงจะได้เลขที่ถูกต้อง โดยเขียนในลักษณะฟังก์ชัน
- เขียนโปรแกรมเพื่อพิมพ์กรอบรูป โดยรับข้อมูลขนาดคอลัมน์และจำนวนบรรทัดของกรอบรูปนั้น โดยเขียนแยกเป็นฟังก์ชันเพื่อการพิมพ์กรอบส่วนที่เป็นบรรทัดแรกกับบรรทัดสุดท้าย และฟังก์ชันส่วนที่เป็นการพิมพ์บรรทัดตรงกลาง เช่น

Enter column : 10

Enter row : 2

Output :

```

 * * * * *
บรรทัดแรกและบรรทัดสุดท้าย { * * } บรรทัดตรงกลาง
พิมพ์ด้วยฟังก์ชัน printHead() { * * } พิมพ์ด้วยฟังก์ชัน printBody()
 * * * * *

```

6. เขียนโปรแกรมเพื่อทำการคำนวณค่าผลต่างของตัวเลข 2 จำนวน คือ A และ B โดยเขียนฟังก์ชันเพื่อทำการคำนวณผลต่างนี้ กำหนดให้ถ้า A มีค่ามากกว่า B ผลลัพธ์ที่ได้จะเท่ากับ A-B นอกจากนั้นผลลัพธ์ที่ได้จะเท่ากับ B-A
7. เขียนโปรแกรมเพื่อพิมพ์ค่าของ y โดยที่  $y = 4x^2 + 3x + 2$  สำหรับค่า x ที่เท่ากับ 1, 3, 5, 7, ..., 21
8. เขียนโปรแกรมเพื่อทำการหาผลคูณของเลขคู่ N จำนวน (นับเลข 2 เป็นตัวที่ 1) โดยรับค่า N จากผู้ใช้ เช่น N = 3 จะทำการคำนวณ  $2 * 4 * 6$  ผลลัพธ์ที่ได้คือ 48 กำหนดโปรโตไทป์ของฟังก์ชันเพื่อหาผลคูณ คือ

```
long cal (int);
```

เมื่อข้อมูลที่ส่งเข้าฟังก์ชันคือ N และฟังก์ชันคืนค่าของผลคูณที่หาได้

9. เขียนโปรแกรมเพื่อทำการหาผลบวกของอนุกรม โดยกำหนดให้รับค่าตัวเลขเริ่มต้นของอนุกรม และตัวเลขสุดท้ายของอนุกรม จากนั้นนำตัวเลขทั้งหมดในอนุกรมนั้นมาบวกกัน สมมติ ตัวเลขเริ่มต้น คือ 5 ตัวเลขตัวสุดท้ายคือ 10 จะทำการนำตัวเลขตั้งแต่ 5 – 10 มาบวกกัน ผลลัพธ์ที่ได้คือ 45 เป็นต้น กำหนดโปรโตไทป์เพื่อหาผลรวม คือ

```
long cal (int, int);
```

โดยที่ข้อมูลเข้าฟังก์ชันตัวแรกคือเลขเริ่มต้น ข้อมูลเข้าฟังก์ชันตัวที่ 2 คือ เลขที่สิ้นสุด และคืนค่าผลรวมของอนุกรม

10. ให้เขียนโปรแกรมเพื่อแสดงผลลัพธ์ของการแปลงเลขฐานสิบ ให้เป็นเลขฐาน 8 และฐาน 16 ตามลำดับ โดยให้รับเลขตั้งต้น และเลขตัวสุดท้ายที่ต้องการให้แสดงผลจากผู้ใช้

กำหนดให้แสดงผลลัพธ์ดังนี้ หากผู้ใช้ป้อนเลขตั้งต้น คือ 20 และเลขสุดท้ายคือ 30

| Dec | Oct | Hex |
|-----|-----|-----|
| 20  | 24  | 14  |
| 21  | 25  | 15  |
| ... | ... | ... |
| 30  | 36  | 1E  |

11. เขียนโปรแกรมเพื่อคำนวณปริมาตรที่ใช้ในการทาห้อง 1 ห้อง โดยให้ผู้ใช้ระบุความกว้าง ความยาว และความสูงของห้อง มีหน่วยเป็นฟุต (ทั้งนี้ยังไม่ต้อง พิจารณาแยกพื้นที่ของประตู หน้าต่าง - ให้คิดรวมกัน) กำหนดให้สี 1 แกลลอนทาพื้นที่ได้ 250 ตร.ฟุต
12. เขียนโปรแกรมเพื่อรับข้อมูลความยาวด้าน 3 ด้านของรูปสามเหลี่ยม และทดสอบว่าด้านทั้ง 3 ที่ผู้ใช้ป้อนเข้าสู่ระบบเป็นด้านของ รูปสามเหลี่ยมเดียวกันหรือไม่ (วิธีตรวจสอบ ผลรวมของด้าน 2 ด้านของสามเหลี่ยมต้องมากกว่าด้านที่ 3 เสมอ)
13. เขียนโปรแกรมเพื่อทำการคำนวณผลบวกของเลขคู่ N จำนวน โดยกำหนดให้เขียนฟังก์ชันเพื่อทำการคำนวณดังกล่าว ตัวอย่างการทำงานของฟังก์ชันนี้ เช่น  $cal(5) = 25$   
 $cal(2) = 3$
14. เขียนโปรแกรมเพื่อตรวจสอบว่าเลขจำนวนเต็มที่ได้รับค่าเข้ามาเป็นจำนวนเฉพาะหรือไม่ (จำนวนเฉพาะ คือ เลขจำนวนเต็มบวกที่มากกว่า 1 ที่นอกจาก 1 และตัวมันเองแล้วไม่มีเลขจำนวนใดหารลงตัว) โดยกำหนดให้เขียนฟังก์ชันเพื่อทำการตรวจสอบดังกล่าว
15. เขียนโปรแกรมเพื่อทำการรับค่าตัวแปรจำนวนเต็มบวก จากนั้นทำการสลับตำแหน่งของเลขจำนวนนั้น กำหนดให้เลขที่รับเข้ามามีค่าอยู่ในช่วงเลขจำนวนเต็มมีค่าไม่เกิน 9999 โดยขั้นตอนของการสลับค่าให้เขียนฟังก์ชันเพื่อเรียกใช้งาน กำหนดให้ชื่อฟังก์ชัน reverse(n) ซึ่งมีหลักการทำงานดังนี้

ตัวอย่าง  $reverse(1234) = 4321$

$reverse(223) = 322$  เป็นต้น

16. เขียนโปรแกรมเพื่อแสดงตารางสูตรคูณดังตัวอย่างข้างล่าง โดยให้ผู้ใช้กำหนดขอบเขตของแม่สูตรคูณ เช่นผู้ใช้ป้อน 4 และ 3 ให้แสดงผลดังตัวอย่าง

|   |   |   |   |    |
|---|---|---|---|----|
| x | 1 | 2 | 3 | 4  |
| 1 | 1 | 2 | 3 | 4  |
| 2 | 2 | 4 | 6 | 8  |
| 3 | 3 | 6 | 9 | 12 |

17. เขียนโปรแกรมเพื่อทำการรับค่าตัวเลขจำนวนเต็มบวกค่าหนึ่ง จากนั้นทำการนับจำนวนของตัวเลข 0-9 ที่ผู้ใช้ระบุซึ่งปรากฏในตัวเลขจำนวนเต็มบวกนั้น โดยมีค่าไม่เกิน 9999 กำหนดให้เขียนฟังก์ชัน  $\text{count}(n,k)$  เพื่อทำหน้าที่ในการนับดังกล่าว ซึ่งมีหลักการทำงานดังนี้

$$\text{ตัวอย่าง } \text{count}(4214,4) = 2$$

$$\text{count}(73,5) = 0$$

18. เขียนโปรแกรมเพื่อทำการพิมพ์ N เทอมแรกของอนุกรม Fibonacci ซึ่งกำหนดให้ 2 เทอมแรกมีค่าเท่ากับ 1,1 (อนุกรม Fibonacci หมายถึง อนุกรมที่มีแต่ละเทอมเป็นค่าผลบวกของ 2 เทอมที่อยู่ก่อน เช่น 1,1,2,3,5,8,13,...) โดยกำหนดให้เขียนฟังก์ชันเพื่อการพิมพ์ผลลัพธ์ดังกล่าว ให้เขียนในลักษณะ Recursive และเขียนในลักษณะธรรมดา
19. เขียนโปรแกรมเพื่อคำนวณรายได้ของเซลแมนคนหนึ่ง โดยรับข้อมูลรหัสพนักงาน ยอดขาย ต้นทุน และให้แสดงรายงานต่าง ๆ คือ รหัสพนักงาน ยอดขาย ต้นทุน กำไร (= ยอดขาย - ต้นทุน) และค่าคอมมิชชัน แสดงดังตัวอย่างข้างล่าง กำหนดว่าหากยอดขายมีมูลค่าไม่เกิน 5000 ให้คิดค่าคอมมิชชัน 10 เปอร์เซ็นต์ของกำไร แต่หากเกินกว่านั้นให้คิดคอมมิชชันที่ 5 เปอร์เซ็นต์ของกำไร ตัวอย่างผลลัพธ์การทำงาน

| Number | Sales  | Value  | Profit | Commission |
|--------|--------|--------|--------|------------|
| -----  | -----  | -----  | -----  | -----      |
| 1234   | 234.56 | 174.56 | 60.00  | 6.00       |
| -----  | -----  | -----  | -----  | -----      |

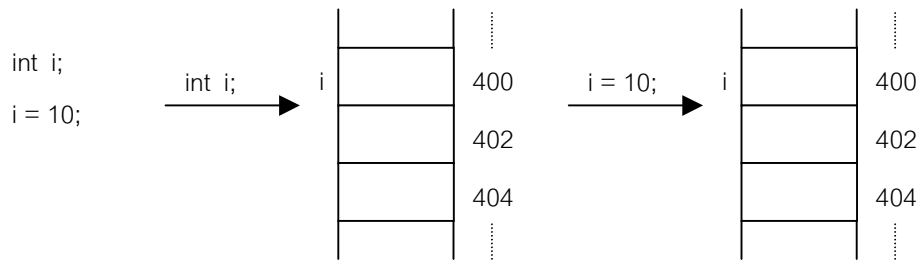
# พอยน์เตอร์ ( Pointer )

# 5

พอยน์เตอร์เป็นชนิดข้อมูลชนิดหนึ่งในภาษาซี ซึ่งมีความเร็วในการทำงานสูงและช่วยให้การเขียนภาษาซีมีความยืดหยุ่น การใช้งานพอยน์เตอร์ค่อนข้างซับซ้อน แต่หากมีความเข้าใจดีแล้ว พอยน์เตอร์จะเป็นจุดเด่นอย่างหนึ่งในการเขียนภาษาซี

## 1. พอยน์เตอร์กับแอดเดรส (Pointers and Addresses)

การประกาศตัวแปร เช่น `int i;` หากพิจารณาคำสั่งดังกล่าวจะเป็นการประกาศ (Declaration) ตัวแปรชื่อ `i` เป็นตัวแปรประเภท `int` จะแสดงภาพจำลองการแทนข้อมูลในหน่วยความจำได้ดังรูปที่ 5.1

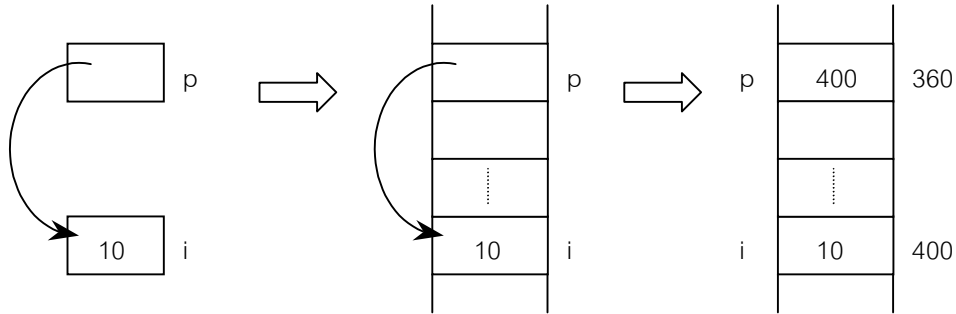


รูปที่ 5.1 แสดงการแทนข้อมูลในหน่วยความจำของตัวแปรประเภทพื้นฐาน

เมื่อมีการประกาศตัวแปรจะมีการจองพื้นที่หน่วยความจำที่มีขนาดเท่ากับประเภทของตัวแปรนั้น การอ้างอิงตัวแปรจะเป็นการอ้างถึงแอดเดรสหรือตำแหน่งที่อยู่ในหน่วยความจำ ในที่นี้สมมติว่ามีการจองพื้นที่ที่แอดเดรส 400 ให้กับตัวแปร `i` (ขนาดของตัวแปรแต่ละประเภทในภาษาซีขึ้นอยู่กับบริษัทผู้ผลิตคอมพิวเตอร์ สามารถตรวจสอบได้จากคู่มือ) เมื่อมีการนำตัวแปรนั้นมาใช้งาน เช่น คำสั่ง `i = 10;` จะเป็นการกำหนดให้ `i` มีค่าเท่ากับ 10 จะมีกระบวนการเก็บค่านั้นลงในพื้นที่หน่วยความจำที่จองไว้ ในที่นี้คือแอดเดรส 400 ดังรูปที่ 5.1

พอยน์เตอร์เป็นชนิดข้อมูลชนิดหนึ่งที่แตกต่างจากชนิดข้อมูลพื้นฐานอื่น ๆ เนื่องจากตัวแปรพอยน์เตอร์เป็นตัวแปรที่ใช้เก็บค่าแอดเดรสของตัวแปรอื่น ๆ หากมีตัวแปร `i` เป็นตัวแปรประเภท `int` และตัวแปร `p` เป็นตัวแปรประเภทพอยน์เตอร์ที่เก็บค่าแอดเดรสของตัวแปร `i` (หรือ `p` ชี้ไปที่ตัวแปร `i`) จะสามารถจำลองการแทนข้อมูลในหน่วยความจำดังรูปที่ 5.2

สมมติให้ตัวแปร  $p$  อยู่ที่แอดเดรส 360 ตัวแปร  $p$  ชี้ไปที่ตัวแปร  $i$  หมายถึง ค่าที่เก็บในตัวแปร  $p$  จะเป็นแอดเดรสของตัวแปร  $i$  เพราะฉะนั้น  $p$  จะเก็บค่า 400 ซึ่งเป็นแอดเดรสของ  $i$  ในขณะที่  $i$  จะเก็บค่าประเภท `int` มีค่าเท่ากับ 10



รูปที่ 5.2 แสดงการแทนข้อมูลในหน่วยความจำของตัวแปรประเภทพอยน์เตอร์

## 2. การประกาศตัวแปรพอยน์เตอร์

การประกาศตัวแปรพอยน์เตอร์จะใช้การดำเนินการชนิดเอกภาค (Unary Operator) `*` ซึ่งมีชื่อเรียกว่า เป็นภาษาอังกฤษว่า Indirection หรือ Dereferencing Operator มีรูปแบบคำสั่งดังนี้

```
int *ip;
```

เป็นการประกาศตัวแปร `ip` ให้เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรชนิด `int` หากเป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรชนิดอื่น จะต้องประกาศชนิดของตัวแปรพอยน์เตอร์ให้สอดคล้องกับชนิดของตัวแปรนั้นเท่านั้น (ยกเว้นตัวแปรพอยน์เตอร์ชนิด `void` ที่สามารถชี้ไปยังตัวแปรชนิดใดก็ได้) เช่น

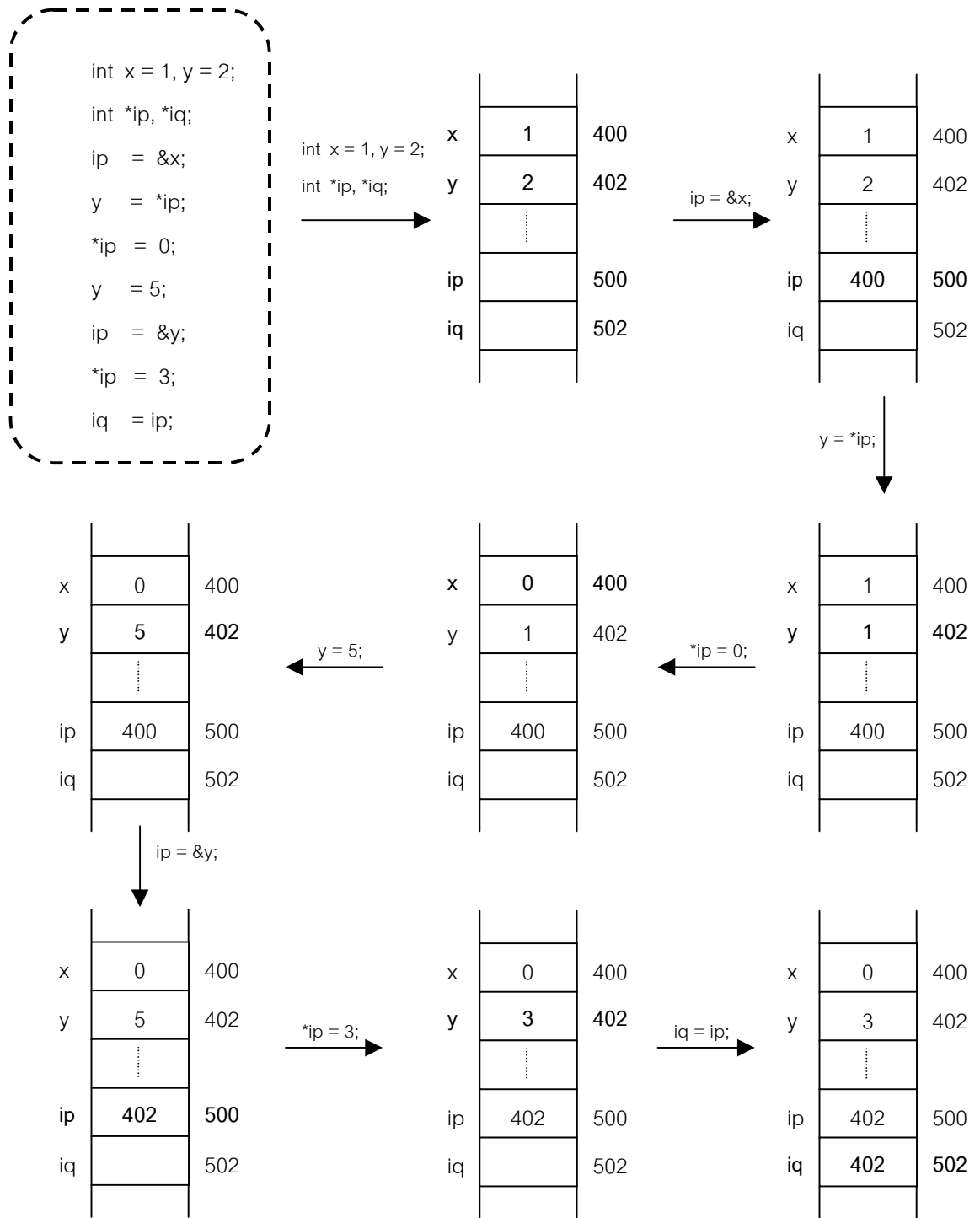
```
float *dp, atof(char *);
```

เป็นการประกาศตัวแปร `dp` เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรประเภท `float` และประกาศฟังก์ชัน `atof` มีพารามิเตอร์เป็นตัวแปรพอยน์เตอร์ชนิด `char` และมีการคืนค่ากลับเป็น `float` คำสั่งดังกล่าวสามารถแยกเป็น 2 คำสั่งได้แก่

```
float *dp;
float atof(char *);
```

## 3. การกำหนดค่าและการอ่านค่าตัวแปรพอยน์เตอร์

การกำหนดค่าให้กับตัวแปรพอยน์เตอร์จะเป็นการกำหนดแอดเดรสของตัวแปรที่มีชนิดข้อมูลสอดคล้องกับชนิดข้อมูลของตัวแปรพอยน์เตอร์เท่านั้น โดยการใช้ การดำเนินการชนิดเอกภาค (Unary Operator) `&` เป็นตัวดำเนินการที่อ้างถึงแอดเดรสของสิ่งใด ๆ ดังรูปที่ 5.3



รูปที่ 5.3 แสดงการกำหนดค่าและการอ่านค่าตัวแปรพอยน์เตอร์

จากรูปที่ 5.3 สามารถอธิบายการทำงานของแต่ละคำสั่งได้ดังนี้

- `int x = 1, y = 2;`

เป็นการประกาศตัวแปร `x` และ `y` โดยกำหนดให้มีชนิดเป็น `int` รวมทั้งกำหนดค่าเริ่มต้นให้กับตัวแปร `x` ให้มีค่าเท่ากับ 1 และกำหนดค่าเริ่มต้นให้ตัวแปร `y` มีค่าเท่ากับ 2

.....

- `int *ip, *iq;`

เป็นการประกาศตัวแปร `ip` และ `iq` เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรประเภท `int`

.....

- `ip = &x;`

เป็นการกำหนดค่าแอดเดรสของตัวแปร `x` ให้กับตัวแปรพอยน์เตอร์ `ip` (อ่านว่า `ip` ชี้ไปยัง `x`) เครื่องหมาย `&` จะใช้กับออปเจ็คที่มีการเก็บอยู่ในหน่วยความจำ เช่น ตัวแปรชนิดต่าง ๆ หรือสมาชิกภายในอาเรย์เท่านั้น จะไม่สามารถใช้กับนิพจน์ ค่าคงที่ และตัวแปรประเภทรีจิสเตอร์ (register) ได้

.....

- `y = *ip;`

เป็นการกำหนดให้ตัวแปร `y` มีค่าเท่ากับค่าที่เก็บอยู่ที่แอดเดรสที่ตัวแปร `ip` เก็บอยู่ ในที่นี้ `ip` ชี้ไปยังตัวแปร `x` ซึ่งมีค่า 1 เพราะฉะนั้นจะเป็นการกำหนดค่า 1 ให้แก่ตัวแปร `y` ภายในตัวแปรพอยน์เตอร์ `ip` จะเก็บค่าแอดเดรสของตัวแปร `x`

ข้อควรระวัง! หากใช้คำสั่ง

```
y = ip;
```

เป็นการกำหนดให้ `y` มีค่าเท่ากับค่าที่เก็บอยู่ที่ตัวแปร `ip` ซึ่งได้แก่แอดเดรสของตัวแปร `x` ซึ่งผิดวัตถุประสงค์และผิดรูปแบบการเขียนภาษาซี

.....

- `*ip = 0;`

เป็นการกำหนดค่าผ่านตัวแปรพอยน์เตอร์ โดยทำการกำหนดค่า 0 ให้แก่ตัวแปรที่ `ip` ชี้อยู่ ในที่นี้ `ip` ชี้อยู่ที่ตัวแปร `x` จึงเป็นการกำหนดค่าให้ตัวแปร `x` มีค่าเท่ากับ 0

.....

- `y = 5;`

เป็นการกำหนดให้ตัวแปร `y` มีค่าเท่ากับ 5

.....

- `ip = &y;`

ทำการเปลี่ยนให้ตัวแปรพอยน์เตอร์ชี้ไปยังตัวแปรตัวอื่นที่มีชนิดสอดคล้องกับตัวแปรพอยน์เตอร์ จากเดิมที่ตัวแปรพอยน์เตอร์ `ip` ชี้ไปยัง `x` จะเปลี่ยนใหม่เป็นชี้ไปยังตัวแปร `y`

- `*ip = 3;`

เป็นการกำหนดค่าให้กับตัวแปรที่พอยน์เตอร์ `ip` ชี้อยู่ ซึ่ง ณ ขณะนี้คือตัวแปร `y` จะเป็นการกำหนดค่าให้กับตัวแปร `y` ให้เท่ากับ 3 การทำงานในขั้นตอนนี้จะไม่มีผลใด ๆ กับตัวแปร `x`

- `iq = ip;`

เป็นการกำหนดค่าตัวแปรพอยน์เตอร์หนึ่งให้กับตัวแปรพอยน์เตอร์อีกตัวหนึ่งได้โดยไม่ต้องใช้ Dereferencing Operator (เครื่องหมาย \*)

ในที่นี้เป็นการกำหนดค่าที่เก็บอยู่ใน `ip` ให้กับ `iq` จะได้ว่า `iq` ชี้ไปยังตัวแปรที่ `ip` ชี้อยู่ ในที่นี้คือตัวแปร `y` ในคำสั่งจะเป็นการนำค่าแอดเดรสที่เก็บอยู่ใน `ip` ไปกำหนดให้แก่ `iq` หลังจากนั้นจะสามารถนำตัวแปรพอยน์เตอร์ `iq` ไปใช้ได้เหมือนตัวแปรพอยน์เตอร์ตัวหนึ่ง

นอกจากนี้ยังสามารถใช้เครื่องหมาย \* ในนิพจน์รูปแบบอื่น ๆ เช่น

```
*ip = *ip + 10; /* เป็นการเพิ่มค่าให้กับตัวแปรที่ ip ชี้อยู่อีก 10 */
```

```
y = *ip + 1; /* เป็นการนำค่าของตัวแปรที่ ip ชี้อยู่บวก 1 */
```

```
/* แล้วกำหนดให้ตัวแปร y */
```

```
*ip += 1; หรือ ++*ip; หรือ (*ip)++
```

เป็นการเพิ่มค่าให้กับตัวแปรที่ `ip` ชี้อยู่ขึ้น 1 ในนิพจน์สุดท้ายจะต้องใส่วงเล็บครอบ `*ip` เพื่อเป็นการเพิ่มค่าให้กับตัวแปรที่ `ip` ชี้อยู่ เนื่องจากการทำงานจะทำจากขวามาซ้าย มิฉะนั้นจะเป็นการเพิ่มค่าที่แอดเดรสที่อยู่ใน `ip` ขึ้น 1 แล้วจึงอ้างถึงข้อมูลในแอดเดรสใหม่ ซึ่งจะทำให้ได้ค่าที่ผิดพลาด

เราสามารถทำการกำหนดค่าให้แก่ตัวแปรพอยน์เตอร์ในช่วงของการประกาศตัวแปรก็สามารถทำได้ดังตัวอย่าง

```
float score, *pScore=&score;
```



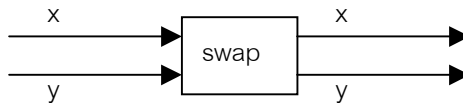
#### 4. พอยน์เตอร์และอาร์กิวเมนต์ของฟังก์ชัน

เนื่องจากภาษาซีมีการส่งอาร์กิวเมนต์ (argument) ให้กับฟังก์ชันแบบ By Value และฟังก์ชันสามารถคืนค่า (return) ค่าได้เพียงหนึ่งค่า หากต้องการให้ฟังก์ชันมีการเปลี่ยนแปลงค่าและคืนค่ากลับมายังฟังก์ชันที่เรียกใช้มากกว่าหนึ่งค่าจะต้องนำพอยน์เตอร์เข้ามาช่วย

ตัวอย่างเช่น หากต้องการเขียนฟังก์ชันเพื่อสลับค่าของตัวแปร 2 ตัว ผลลัพธ์ที่ต้องการได้จากฟังก์ชันนี้จะมีค่าของตัวแปร 2 ตัวที่ทำการสลับค่ากัน หากเขียนฟังก์ชันทั่วไปจะไม่สามารถแก้ปัญหานี้ได้ เนื่องจากฟังก์ชันแต่ละฟังก์ชันจะคืนค่าคำตอบได้เพียงค่าเดียว จึงต้องใช้พอยน์เตอร์เข้ามาช่วย โดยการส่งค่าแอดเดรสของตัวแปรทั้ง 2 ให้กับฟังก์ชันที่จะสลับค่าของตัวแปรทั้ง 2 ผ่านทางตัวแปรพอยน์เตอร์ที่เป็นพารามิเตอร์ของฟังก์ชัน ดังตัวอย่างที่ 5.1

**ตัวอย่างที่ 5.1** โปรแกรมการสลับค่าตัวแปร 2 ตัวโดยผ่านฟังก์ชัน จะแสดงการรับพารามิเตอร์เป็นตัวแปรชนิดพอยน์เตอร์

##### วิเคราะห์



จากการวิเคราะห์พบว่าในการเรียกใช้งานฟังก์ชัน swap() จะมีการรับข้อมูลเข้า 2 ค่า และเมื่อทำงานเสร็จจะมีการส่งข้อมูลกลับให้ฟังก์ชันที่เรียกใช้งานจำนวน 2 ค่า ซึ่งฟังก์ชันลักษณะนี้ต้องใช้ฟังก์ชันที่รับพารามิเตอร์เป็นตัวแปรชนิดพอยน์เตอร์

```

#include <stdio.h>
void swap (int *, int *);
void main () {
 int x = 5, y = 10;
 printf("Before swap : x = %d y = %d\n", x, y);
 swap (&x, &y); /* Pass address of x and y to swap() */
 printf("After swap : x = %d y = %d\n", x, y);
}

```

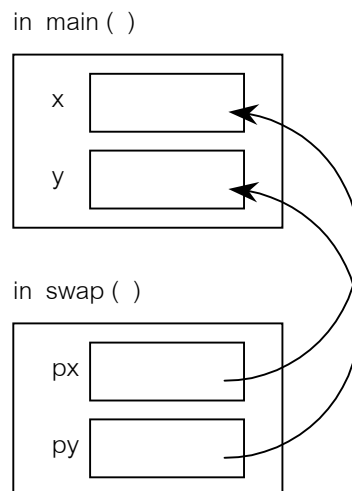
```

void swap (int *px, int *py) {
 int temp;
 temp = *px; /* Keep x value to temp */
 *px = *py; /* Assign y value to x */
 py = temp; / Assign old x value to y */
}

```

---

อาร์กิวเมนต์ที่เป็นประเภทพอยน์เตอร์จะช่วยให้ฟังก์ชันสามารถเปลี่ยนค่าให้กับตัวแปรที่ส่งเข้ามาได้ เนื่องจากอาร์กิวเมนต์นั้นจะเก็บแอดเดรสของตัวแปรที่ส่งเข้ามา เมื่อมีการเปลี่ยนแปลงค่าของอาร์กิวเมนต์ผ่านตัวดำเนินการ \* (Dereferencing Operator) ค่าของตัวแปรที่ส่งเข้ามาจะถูกเปลี่ยนค่าพร้อมกันในทันที แสดงความสัมพันธ์ของอาร์กิวเมนต์กับฟังก์ชันดังรูปที่ 5.4



รูปที่ 5.4 แสดงความสัมพันธ์ของการส่งอาร์กิวเมนต์แบบพอยเตอร์กับฟังก์ชัน

รูปแบบโปรโตไทป์ของการส่งอาร์กิวเมนต์แบบพอยน์เตอร์ให้กับฟังก์ชัน

```
void swap (int *, int *);
```

ตัวแปรใดที่ต้องการรับค่าอาร์กิวเมนต์แบบพอยน์เตอร์ให้เพิ่มเครื่องหมาย \* ตามหลังการประกาศชนิดของตัวแปร

รูปแบบการเรียกใช้งานฟังก์ชันที่มีการรับอาร์กิวเมนต์แบบพอยน์เตอร์

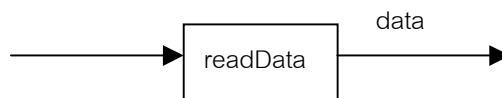
```
swap (& x, &y);
```

ตัวแปรใดที่ต้องการส่งค่าอาร์กิวเมนต์แบบพอยน์เตอร์ให้เพิ่มเครื่องหมาย & นำหน้าชื่อตัวแปรนั้น

ตัวแปรพอยน์เตอร์เป็นที่นิยมในฟังก์ชันทั่วไปเนื่องจากมีความเร็วในการทำงานสูงกว่าตัวแปรทั่วไปและประหยัดทรัพยากรของระบบมากกว่า โดยเฉพาะการส่งตัวแปรที่มีขนาดใหญ่ไปยังฟังก์ชัน หากใช้ตัวแปรชนิดพอยน์เตอร์มาช่วยจะเห็นความแตกต่างได้มากกว่า ตัวอย่างเช่นหากมีการส่งตัวแปรชนิด float ไปยังฟังก์ชัน ๗ หนึ่งกระบวนการที่เกิดขึ้นคือ มีการจองพื้นที่ให้กับพารามิเตอร์ในฟังก์ชันนั้นมีขนาดเท่ากับ float (8 ไบต์) และจะต้องมีการสำเนาค่าตัวแปรจากฟังก์ชันที่เรียกใช้ไปให้กับพารามิเตอร์ในฟังก์ชันนั้น แต่หากมีการใช้พารามิเตอร์ในฟังก์ชันเป็นตัวแปรชนิดพอยน์เตอร์ กระบวนการที่เกิดขึ้นคือ มีการจองพื้นที่ให้กับตัวแปรชนิดพอยน์เตอร์ด้วยขนาดที่ใช้เก็บค่าแอดเดรสได้ (ไม่ว่าจะเป็นพอยน์เตอร์ชี้ไปยังตัวแปรประเภทใดจะมีขนาดเท่ากันเสมอ) และสำเนาแอดเดรสจากอาร์กิวเมนต์ในฟังก์ชันที่เรียกใช้มายังพารามิเตอร์ในฟังก์ชัน

**ตัวอย่างที่ 5.2** โปรแกรมเพื่อรับข้อมูลจำนวนจริง 1 จำนวนจากผู้ใช้ โดยเปรียบเทียบวิธีการเขียนฟังก์ชันที่ส่งค่ากลับแบบ By value และการส่งค่ากลับโดยใช้อาร์กิวเมนต์แบบพอยน์เตอร์

#### วิเคราะห์



จากการวิเคราะห์พบว่าในการเรียกใช้งานฟังก์ชัน `readData ( )` ไม่จำเป็นต้องมีการรับข้อมูลเข้า และเมื่อทำงานเสร็จจะมีการส่งข้อมูลกลับให้ฟังก์ชันที่เรียกใช้งานจำนวน 1 ค่า ซึ่งฟังก์ชันลักษณะนี้สามารถเขียนได้ทั้งสองลักษณะ

การส่งค่ากลับแบบ By value

```

#include <stdio.h>
1. int readData();
2. void main()
3. { int x;
4. x= readData();
5. }
6. int readData(){
7. int data;
8. printf("Enter value :");
9. scanf("%d", &data);
10. return (data);
11. }

```

การส่งค่ากลับโดยใช้อาร์กิวเมนต์แบบพอยน์เตอร์

```

#include <stdio.h>
void readData(int *);
void main()
{ int x;
 readData(&x);
}
void readData(int *data){
 printf("Enter value :");
 scanf("%d", data);
}

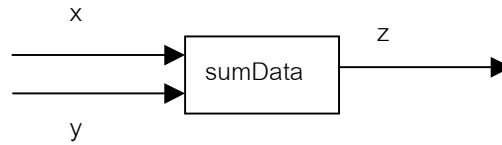
```

ข้อแตกต่างที่สำคัญของการส่งค่ากลับโดยใช้อาร์กิวเมนต์แบบพอยน์เตอร์กับการเขียนโปรแกรมในรูปแบบการส่งค่ากลับแบบ By value จากตัวอย่าง 5.2 คือ

- รูปแบบของโปรโตไทป์ ( บรรทัดที่ 1 )
- คำสั่งในการเรียกใช้งานฟังก์ชัน readData( ) โดยฟังก์ชัน main ( ) ( บรรทัดที่ 4 )
- การประกาศหัวของฟังก์ชัน ( บรรทัดที่ 6 )
- การใช้งานอาร์กิวเมนต์แบบพอยน์เตอร์ไม่จำเป็นต้องประกาศตัวแปรใหม่( บรรทัดที่ 7 )
- คำสั่งในการอ่านข้อมูลตำแหน่งหน้าตัวแปร data ในคำสั่ง scanf ไม่มีเครื่องหมาย & ( บรรทัดที่ 9 )
- การใช้งานอาร์กิวเมนต์แบบพอยน์เตอร์การส่งค่ากลับไม่ต้องใช้คำสั่ง return( ) ( บรรทัดที่ 10 )

**ตัวอย่างที่ 5.3** โปรแกรมเพื่อหาผลบวกจำนวนจริง 2 จำนวน โดยเปรียบเทียบวิธีการเขียนฟังก์ชันที่ส่งค่ากลับแบบ By value และการส่งค่ากลับโดยใช้อาร์กิวเมนต์แบบพอยน์เตอร์

#### วิเคราะห์



จากการวิเคราะห์พบว่าในการเรียกใช้งานฟังก์ชัน `sumData()` จะมีการรับข้อมูลเข้า 2 ค่า และเมื่อทำงานเสร็จจะมีการส่งข้อมูลกลับให้ฟังก์ชันที่เรียกใช้งานจำนวน 1 ค่า ซึ่งฟังก์ชันลักษณะนี้สามารถเขียนได้ทั้งสองลักษณะ

#### การส่งค่ากลับแบบ By value

```

#include <stdio.h>
1. int sumData(int , int);
2. void main()
3. { int x , y;
4. x = 3;
5. y = 2;
6. z=sumData(x,y);
7. printf("The result is %d", z);
8. }
9. int sumData(int x , int y){
10. int z;
11. z = x + y;
12. return (z);
13. }
```

#### การส่งค่ากลับโดยใช้อาร์กิวเมนต์แบบพอยน์เตอร์

```

#include <stdio.h>
void sumData(int , int , int *);
void main()
{ int x , y;
 x = 3;
 y = 2;
 sumData(x,y, &z);
 printf("The result is %d", z);
}
void sumData(int x , int y , int * z){
 *z = x + y;
}
```

ข้อแตกต่างที่สำคัญของการส่งค่ากลับโดยใช้อาร์กิวเมนต์แบบพอยน์เตอร์กับการเขียนโปรแกรมในรูปแบบการส่งค่ากลับแบบ By value จากตัวอย่าง 5.3 คือ

- รูปแบบของโปรโตไทป์ ( บรรทัดที่ 1 )
- คำสั่งในการเรียกใช้งานฟังก์ชัน sumData ( ) โดยฟังก์ชัน main ( ) ( บรรทัดที่ 6 )
- การประกาศหัวของฟังก์ชัน ( บรรทัดที่ 9 )
- การใช้งานอาร์กิวเมนต์แบบพอยน์เตอร์ไม่จำเป็นต้องประกาศตัวแปรใหม่( บรรทัดที่ 10 )
- การอ้างถึงตัวแปร z ซึ่งเป็นผลลัพธ์ ในการใช้งานตัวแปรพอยน์เตอร์จะต้องอ้างด้วย \*z ( บรรทัดที่ 11 )
- การใช้งานอาร์กิวเมนต์แบบพอยน์เตอร์การส่งค่ากลับไม่ต้องใช้คำสั่ง return( ) ( บรรทัดที่ 12 )

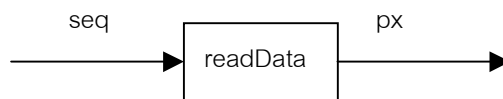
**ตัวอย่างที่ 5.4** เขียนโปรแกรมเพื่อรับข้อมูลจำนวนจริง 3 จำนวนจากผู้ใช้และหาค่าเฉลี่ยของค่าที่รับเข้ามาทั้งหมด โดยเขียนในลักษณะการส่งอาร์กิวเมนต์แบบพอยน์เตอร์

#### วิเคราะห์

ในการออกแบบโปรแกรมนี้ สามารถแบ่งออกเป็นงานย่อย ๆ ดังนี้

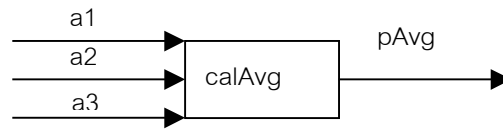
- รับข้อมูล 3 จำนวนจากผู้ใช้
- หาค่าเฉลี่ย
- แสดงผลลัพธ์

จากงานย่อยดังกล่าวสามารถนำมาออกแบบฟังก์ชันย่อยได้ 3 ฟังก์ชันดังนี้

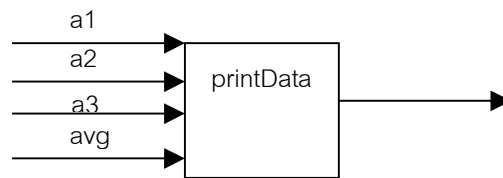


ฟังก์ชัน readData ( ) ทำการรับข้อมูลเลขจำนวนจริงจากผู้ใช้ครั้งละ 1 จำนวน โดยที่ตัวแปร seq จะเป็นตัวแปรที่ชี้แสดงจำนวนครั้งเพื่อแสดงให้ผู้ใช้ทราบว่าขณะนั้นเป็นการป้อนข้อมูลลำดับที่เท่าไร

## วิเคราะห์ (ต่อ)



ฟังก์ชัน `calAvg()` ทำหน้าที่คำนวณหาค่าเฉลี่ย โดยรับค่าตัวเลข 3 จำนวน จากนั้นส่งผลลัพธ์ที่ได้จากการคำนวณกลับคืนให้ฟังก์ชัน `main()`



ฟังก์ชัน `printData()` ทำหน้าที่พิมพ์ค่าตัวเลขทั้ง 3 จำนวนที่รับจากผู้ใช้งานทางจอภาพ รวมทั้งแสดงค่าเฉลี่ยที่ได้จากการคำนวณ

```
#include <stdio.h>
void readData(int, float *);
void calAverage(float, float, float, float *);
void printData(float, float, float, float);
void main() {
 float x1, x2, x3, average;
 readData(1, &x1);
 readData(2, &x2);
 readData(3, &x3);
 calAverage(x1, x2, x3, &average);
 printData(x1, x2, x3, average);
}
void readData(int seq, float *px) {
 printf("Enter value %d : ", seq);
 scanf("%f", px);
}
```

```

void calAverage(float a1, float a2, float a3, float *pAvg) {
 *pAvg = (a1 + a2 + a3)/3.0;
}

void printData(float b1, float b2, float b3, float avg) {
 printf("Average of %.2f, %.2f, %.2f is %.2f", b1, b2, b3, avg);
}

```

**ตัวอย่างที่ 5.5** ใจพทย์เดียวกับตัวอย่างที่ 5.4 คือ เขียนโปรแกรมเพื่อรับข้อมูลจำนวนจริง 3 จำนวนจากผู้  
ใช้และหาค่าเฉลี่ยของค่าที่รับเข้ามาทั้งหมด โดยเขียนในลักษณะพอยน์เตอร์ แต่เขียนในอีกลักษณะหนึ่ง

#### วิเคราะห์

- ในการออกแบบการทำงานของฟังก์ชันต่างๆ จะเหมือนกับในตัวอย่างที่ 5.4 ทุก  
ประการ ยกเว้นในฟังก์ชัน readData()



ฟังก์ชัน readData () ทำการรับข้อมูลเลขจำนวนจริงจากผู้ใช้ครั้งละ 3 จำนวน

- ในเขียนโปรแกรมของฟังก์ชันต่างๆ การส่งค่าให้กับฟังก์ชันในตัวอย่างนี้จะแตกต่างจากตัวอย่างที่ 5.4 โดยในตัวอย่าง 5.4 ตัวแปรที่ทำหน้าที่เป็นข้อมูลเข้าอย่าง  
เดียวจะส่งค่าแบบ By value แต่ตัวใดที่เป็นข้อมูลออกจะส่งค่าแบบพอยน์เตอร์  
แต่สำหรับตัวอย่างนี้ทั้งตัวแปรรับเข้าและตัวแปรส่งออกต่างกำหนดให้เป็นตัวแปร  
แบบพอยน์เตอร์

```

#include <stdio.h>

void readData(float *, float *, float *);

void calAverage(float *, float *, float *, float *);

void printData(float *, float *, float *, float *);

```



```

void main() {
 float x1, x2, x3, average;
 readData(&x1, &x2, &x3);
 calAverage(&x1, &x2, &x3, &average);
 printData(&x1, &x2, &x3, &average);
}

void readData(float *px1, float *px2, float *px3) {
 printf("Enter value 1 : ");
 scanf("%f", px1);
 printf("Enter value 2 : ");
 scanf("%f", px2);
 printf("Enter value 3 : ");
 scanf("%f", px3);
}

void calAverage(float *pa1, float *pa2, float *pa3, float *pAvg) {
 *pAvg = (*pa1 + *pa2 + *pa3)/3.0;
}

void printData(float *pb1, float *pb2, float *pb3, float *pAvg) {
 printf("Average of %.2f, %.2f, %.2f is %.2f", *pb1, *pb2, *pb3, *pAvg);
}

```

---

**ตัวอย่างที่ 5.6** เขียนโปรแกรมเพื่อคำนวณพื้นที่ของสี่เหลี่ยมรูปหนึ่ง โดยรับข้อมูลความกว้างและความยาวของรูปสี่เหลี่ยมจากผู้ใช้งาน กำหนดให้ใช้ฟังก์ชันเพื่อคำนวณพื้นที่ของรูปสี่เหลี่ยมดังโปรแกรมต่อไปนี้

```
void calRecArea(float, float, float *);
```

โดยที่พารามิเตอร์ตัวแรกคือความกว้าง พารามิเตอร์ตัวที่ 2 คือความยาว และพารามิเตอร์ตัวที่ 3 คือพื้นที่ของรูปสี่เหลี่ยม

---

```

#include <stdio.h>
void calRecArea(float , float , float *)

```

```

void main() {
 float width, length, area;
 printf("Enter width : ");
 scanf("%f", &width);
 printf("Enter length : ");
 scanf("%f", &length);
 calRecArea(width, length, &area);
 printf("Rectangle area is %.2f", area);
}

void calRecArea(float w, float l, float *pArea) {
 *pArea = w * l;
}

```

---

**ตัวอย่างที่ 5.7** เขียนโปรแกรมเพื่อรับข้อมูลจำนวนจริง 3 จำนวนจากผู้ใช้และหาค่าเฉลี่ยของค่าที่รับเข้ามาทั้งหมด ให้นำว่ามีเลขจำนวนจริงที่รับเข้ามานั้นมีค่าน้อยกว่าค่าเฉลี่ยกี่จำนวน

---

```

#include <stdio.h>

void readData(int, float *);
void calAverage(float, float, float, float *);
void findLessAverage(float, float, float, float, int *);

void main() {
 float x1, x2, x3, average;
 int num;

 readData(1, &x1);
 readData(2, &x2);
 readData(3, &x3);

 calAverage(x1, x2, x3, &average);
 findLessAverage(x1, x2, x3, average, &num);

 printf("Less than average = %d", num);
}

void readData(int seq, float *px) {
 printf("Enter value %d : ", seq);
 scanf("%f", px);
}

void calAverage(float a1, float a2, float a3, float *pAvg) {
 *pAvg = (a1 + a2 + a3)/3.0;
}

```

```
void findLessAverage(float b1, float b2, float b3, float avg, int *pNum) {
 *pNum = 0;
 if (b1 < avg)
 (*pNum)++;
 if (b2 < avg)
 (*pNum)++;
 if (b3 < avg)
 (*pNum)++;
}
```

---

---

**แบบฝึกหัดบทที่ 5**

1. หาที่ผิดพลาดของคำสั่งต่อไปนี้ และแก้ไขให้ถูกต้อง

```
int a=10, *pa1, *pa2=a;
float f=5.0, *pf1, pf2;
pa1 = a;
pf1 = f;
pf2 = &a;
pf1 = pf2 + pa2;
printf("\nf = %.2f", pf1);
pa2 = &pa1;
printf("\na = %d", pa1);
```

2. หาผลลัพธ์การทำงานของโปรแกรมต่อไปนี้

```
#include <stdio.h>
int x=10;
void func1(int *tx) {
 printf("\nEnter x : ");
 scanf("%d", tx);
 printf("\n1. x = %d, *tx = %d", x, *tx);
}
void func2(int *ta) {
 *ta += x;
 printf("\n2. x = %d, *ta = %d", x, *ta);
}
void func3(int a) {
 x = x - a;
 a = a * 2;
 printf("\n3. x = %d, a = %d", x, a);
}
void main() {
 int x, *px=&x;
 func1(&x); printf("\nx = %d, *px = %d", x, *px);
 func2(px); printf("\nx = %d, *px = %d", x, *px);
 func3(*px); printf("\nx = %d, *px = %d", x, *px);
 func1(px); printf("\nx = %d, *px = %d", x, *px);
 func2(&x); printf("\nx = %d, *px = %d", x, *px);
 func3(x); printf("\nx = %d, *px = %d", x, *px);
}
```

3. อธิบายการทำงานของฟังก์ชันต่อไปนี้

```
void c_out(char *s) {
 while(*s) putchar(*s++);
}
```

4. ข้อต่อไปนี้ข้อใดผิด เพราะอะไร

(ก) int n[ ]; \_\_\_\_\_.

int \*m;

n[0] = 12;

n = m;

\_\_\_\_\_.

(ข) int n=10, arr[n]; \_\_\_\_\_.

(ค) void enterData( ) { \_\_\_\_\_.

int i, data[10]; \_\_\_\_\_.

for (i=0; i<10; i++); \_\_\_\_\_.

printf("Enter data %d", i); \_\_\_\_\_.

scanf("%d", data[i]); \_\_\_\_\_.

} \_\_\_\_\_.

void main( ) { \_\_\_\_\_.

int i; \_\_\_\_\_.

for (i=0; i<10; i++) \_\_\_\_\_.

printf("\Data %d is %d", i, data[i]); \_\_\_\_\_.

} \_\_\_\_\_.

5. ถ้ากำหนดการประกาศตัวแปรไว้ดังนี้

```
float table[10];
```

```
float *pt, *qt;
```

จงอธิบายการทำงานของกาหนดค่าดังต่อไปนี้ (แต่ละข้อย่อยไม่เกี่ยวข้องกัน)

(ก) pt = table;

\*pt = 0;

\*(pt + 2) = 3.14;

(ข) pt = table+2;

qt = pt;

\*qt = 2.178;

(ค) pt = table;

qt = table + 9;

printf("%d", \*qt-\*pt);

(ง) pt = table;

qt = table + 9;

for ( ; pt<qt; pt++)

\*pt = 1.23;

6. เขียนโปรแกรมเพื่อคำนวณเลขยกกำลัง โดยรับค่าเลขฐานและเลขยกกำลังจากผู้ใช้ โดยใช้โปรโตไทป์ที่กำหนด ดังนี้

```
รับข้อมูล void readData (float *, int *);
คำนวณ float power (float *, int);
```

กำหนดให้ข้อมูลที่เข้าสู่ฟังก์ชันทั้ง 2 คือ เลขฐานและเลขยกกำลังตามลำดับ

7. เขียนโปรแกรมเพื่อทำการหาผลบวกของอนุกรม โดยกำหนดให้รับค่าตัวเลขเริ่มต้นของอนุกรม และตัวเลขสุดท้ายของอนุกรม จากนั้นนำตัวเลขทั้งหมดในอนุกรมนั้นมาบวกกัน สมมติ ตัวเลขเริ่มต้น คือ 5 ตัวเลขตัวสุดท้ายคือ 10 จะทำการนำตัวเลขตั้งแต่ 5 – 10 มาบวกกัน ผลลัพธ์ที่ได้คือ 45 เป็นต้น

กำหนดให้เขียนฟังก์ชัน cal() ทำหน้าที่คำนวณหาผลบวกของอนุกรมดังกล่าว ซึ่งมีโปรโตไทป์ดังนี้

```
void cal (int, int, long *);
```

โดยที่ข้อมูลเข้าฟังก์ชันตัวแรก คือ เลขเริ่มต้น ตัวที่ 2 คือ เลขสุดท้าย และตัวสุดท้าย คือ ผลรวม

8. เขียนโปรแกรมเพื่อทำการคำนวณค่าจ้างของพนักงาน โดยกำหนดให้รับค่าจำนวนชั่วโมงการทำงาน (Hour) และประเภทของงานที่พนักงานทำ (Type) โดยมีข้อกำหนดในการคำนวณดังนี้

| ประเภทของงาน | อัตราค่าจ้าง / ชั่วโมง |
|--------------|------------------------|
| 0            | 30                     |
| 1            | 40                     |
| 2            | 45                     |
| 3            | 50                     |

หากผู้ใช้ป้อนประเภทของงานนอกเหนือจากประเภทงานดังกล่าว จะแสดงข้อความว่า "Error Data"

กำหนดให้เขียนฟังก์ชันเพื่อรับข้อมูลประเภทของงาน และอัตราค่าจ้างต่อชั่วโมงไว้ในฟังก์ชันเดียวกัน และเขียนฟังก์ชัน cal() ทำหน้าที่คำนวณค่าจ้างของพนักงาน ซึ่งมีโปรโตไทป์ดังนี้

```
void cal (char , int, float *);
```

โดยที่ข้อมูลเข้าฟังก์ชัน cal() ตัวแรกคือประเภทของงาน ตัวที่ 2 คือ ชั่วโมงทำงาน ตัวที่ 3 คือ ค่าจ้างที่ได้

# ตัวแปรชุด ( Array )

# 6

ตัวแปรชุด ( Arrays ) คือ กลุ่มของข้อมูลที่มีชนิดของข้อมูลเหมือนกัน จึงทำการจัดกลุ่มไว้ด้วยกัน แล้วอ้างถึงด้วยกลุ่มของข้อมูลนั้นด้วยชื่อเดียว และอ้างถึงสมาชิกแต่ละตัวในกลุ่มของตัวแปรชุดนั้นด้วยหมายเลข

ตัวอย่างของปัญหาที่จำเป็นต้องใช้งานตัวแปรชุดเช่น หากต้องการรับข้อมูลคะแนนสอบของนักศึกษาแต่ละคนที่ลงทะเบียนในกระบวนวิชาหนึ่งซึ่งมีนักศึกษาลงทะเบียนเรียน 40 คน ต้องการทราบว่ามีนักศึกษาที่ได้คะแนนต่ำกว่าคะแนนเฉลี่ยกี่คน ในกรณีเช่นนี้จะเห็นว่ามีกรับข้อมูลคะแนนสอบของนักศึกษาแต่ละคน จากนั้นนำมาหาคะแนนเฉลี่ย และจะต้องมีนำข้อมูลคะแนนสอบของนักศึกษาแต่ละคนมาเปรียบเทียบกับคะแนนเฉลี่ยว่าจะมีนักศึกษาที่ได้คะแนนน้อยกว่าคะแนนเฉลี่ยกี่คน การทำงานในลักษณะที่ต้องมีการเก็บข้อมูลที่เหมือน ๆ กันและนำกลับมาใช้เพื่อประมวลผลงานใดงานหนึ่งเช่นนี้ จำเป็นต้องใช้ตัวแปรชุดมาช่วยในการเก็บข้อมูลเสมอ

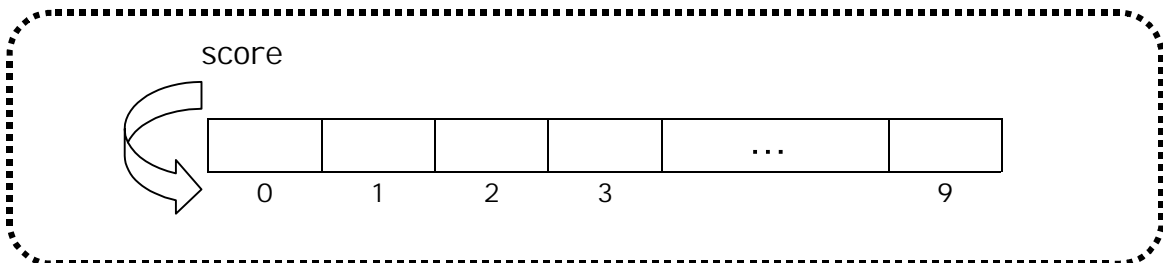
## 1. รูปแบบการประกาศตัวแปรชุด

ในภาษาซีเราสามารถสร้างตัวแปรชุดจากข้อมูลพื้นฐาน อันได้แก่ int float float เป็นต้น รูปแบบของการประกาศตัวแปรชุดทำได้ดังนี้

ชนิดข้อมูล ชื่อตัวแปร [ ขนาดข้อมูล ] ;

ตัวอย่าง 6.1 แสดงการประกาศตัวแปรชุดเพื่อเก็บข้อมูลคะแนนสอบของนักศึกษา 10 คน

```
float score[10];
```



คำสั่ง `float score[10];` เป็นการกำหนดตัวแปรตัวแปรชุดชื่อ `score` เป็นตัวแปรชุดชนิด `float` ที่มีสมาชิกทั้งหมด 10 ตัว ตั้งแต่ `score[0]`, `score[1]`, `score[2]`, ... , `score[9]` **สมาชิกภายในตัวแปรชุดจะเริ่มที่ 0 เสมอ** และสมาชิกตัวสุดท้ายจะอยู่ที่ตำแหน่งของขนาดที่ประกาศไว้ลบด้วย 1 เช่น ประกาศขนาดของตัวแปรชุดไว้ `n` สมาชิกตัวสุดท้ายจะอยู่ที่ตำแหน่ง `n-1`

#### การอ้างถึงสมาชิกแต่ละตัวภายในตัวแปรชุด

การอ้างถึงสมาชิกของตัวแปรชุดจะใช้ระบบดัชนี (Index) โดยผ่านเครื่องหมาย `[ ]` เช่น อ้างถึงสมาชิกตำแหน่งแรกของตัวแปรชุดด้วย `score[0]` เป็นต้น การใช้งานสมาชิกของตัวแปรชุดสามารถใช้งานได้เหมือนตัวแปรพื้นฐานทั่วไป ตัวอย่างต่อไปนี้แสดงคำสั่งที่ใช้งานกับสมาชิกของตัวแปรชุด

คำสั่งในการบวกค่าสมาชิก 3 ตัวแรกของตัวแปรชุด

```
sumThird = score[0] + score[1] + score[2];
```

คำสั่งในการกำหนดค่า 5 ให้กับสมาชิกตัวแรกของตัวแปรชุด

```
score[0] = 5;
```

คำสั่งในการเปรียบเทียบว่าค่าของสมาชิกตัวแรกมากกว่าสมาชิกตัวสุดท้ายหรือไม่

```
if (score[0] > score[9])
 printf ("First is greater than last\n");
```

เราสามารถอ้างถึงสมาชิกทุกตัวภายในตัวแปรชุดอย่างอิสระ ภายในขอบเขตของขนาดที่ได้ประกาศตัวแปรชุดไว้ แต่การใช้ตัวแปรชุดทั่วไปจะเป็นการเข้าถึงสมาชิกโดยใช้ตัวแปรประเภท `int` มาช่วยเป็นดัชนีอ้างถึงสมาชิกที่ต้องการ

ตัวอย่างต่อไปนี้แสดงการเข้าถึงสมาชิกของตัวแปรชุดโดยใช้ตัวแปรประเภท `int` สมมติให้ `i`, `j`, `k` เป็นตัวแปรประเภท `int`

คำสั่งวนซ้ำเพื่อพิมพ์ค่าที่เก็บอยู่ในแต่ละสมาชิกของตัวแปรชุด

```
for (k = 0; k < 10; k++)
 printf ("Value at %d = %d\n", k+1, score[k]);
```



คำสั่งกำหนดให้ค่าสมาชิกตำแหน่งที่  $i+j$  หรือ  $2+3$  คือ สมาชิกตำแหน่งที่ 5 มีค่าเท่ากับ 0

$i = 2;$

$j = 3;$

$\text{score}[i + j] = 0;$

คำสั่งการรับค่าเพื่อนำมาเก็บในตัวแปรตัวแปรชุดสามารถทำได้ด้วยคำสั่ง

```
for (i=0; i < 10; i++) {
 printf("Enter member %d : ", i);
 scanf("%f", &score[i]);
}
```

#### หมายเหตุ

- จากตัวอย่างจะเห็นว่าการอ้างถึงสมาชิกแต่ละตัวภายในตัวแปรชุดจะใช้ลักษณะการอ้างถึงในลักษณะกับการอ้างถึงตัวแปรทั่วไป แต่เมื่อใดที่มีการอ้างถึงแต่ชื่อของตัวแปรตัวแปรชุด เช่น อ้างถึง `score` จะเป็นการอ้างถึงแอดเดรสเริ่มต้นของตัวแปรตัวแปรชุดนั้นโดยอาศัยหลักการเดียวกับตัวแปรชนิดพอยน์เตอร์
- **สิ่งที่ต้องระวัง** คือ ในภาษาซีจะไม่มีการกำหนดให้ตรวจสอบขอบเขตของตัวแปรชุด ผู้เขียนโปรแกรมจะต้องพยายามเขียนโปรแกรมที่เกี่ยวข้องกับสมาชิกของตัวแปรชุดภายในขอบเขตที่ประกาศตัวแปรชุดไว้ หากมีการอ้างถึงสมาชิกตัวแปรชุดนอกขอบเขตที่ได้ระบุไว้ เช่น `score[12]` สิ่งที่ได้คือการไปอ่านข้อมูลในพื้นที่ของหน่วยความจำที่อาจจะเก็บค่าของตัวแปรตัวอื่นหรือเป็นค่าอื่นใดที่ไม่อาจคาดเดาได้

## ตัวอย่างที่ 6.2 ให้รับค่าของจำนวนเต็ม 5 จำนวนจากผู้ใช้ และแสดงผลในลำดับที่กลับกัน

---

```

/* Read five integers from the standard input */
/* and output them in reverse order. */
#include <stdio.h>
#define SIZE 5
main () {
 int k; /* loop control */
 int data[SIZE]; /* data value */
 for (k = 0; k < SIZE; k++)
 scanf ("%d", &data[k]); /* data input */
 for (k = SIZE-1; k >= 0; k--)
 printf ("%d\n", data[k]); /* data output */
}

```

---

จากตัวอย่างจะเป็นการใช้ตัวประมวลผลก่อน (Preprocessor) #define กำหนดให้ SIZE มีค่าเป็น 5 ซึ่งในที่นี้เป็นกรกำหนดค่าคงที่ให้กับขนาดของตัวแปรชุด กระบวนการทำงานของตัวประมวลผลก่อนจะทำงานดังนี้คือ เมื่อสั่งให้มีการแปลคำสั่ง ตัวแปลคำสั่งจะแทนค่า 5 ลงไปทุกที่ที่เจอคำว่า SIZE แล้วจึงทำการคอมไพล์ต่อจนเสร็จเรียบร้อย

การทำงานของตัวอย่างข้างบนจะได้ตัวแปรชุดชื่อ data มีขนาด 5 หลังจากนั้นจะใช้คำสั่งวนซ้ำ for ให้อ่านค่าเข้าทางอุปกรณ์ป้อนข้อมูลมาตรฐาน (Standard Input ปกติจะเป็นคีย์บอร์ด) ด้วยฟังก์ชัน scanf ( ) มาเก็บยังสมาชิกของตัวแปรชุดแต่ละตัวตั้งแต่ตัวแรกคือ data[0] จนถึงตัวสุดท้ายคือ data[4] สังเกตว่าจะต้องส่งแอดเดรสของสมาชิกแต่ละตัวให้กับฟังก์ชัน scanf ( ) ด้วยเครื่องหมาย &

หลังจากนั้นนำข้อมูลที่ได้อามาพิมพ์ออกทางอุปกรณ์แสดงผลมาตรฐาน (Standard Output ปกติจะเป็นจอภาพ) ในลำดับที่กลับกัน คือ จากตัวสุดท้ายคือ data[4] จนถึงตัวแรกคือ data[0]

สมาชิกของตัวแปรชุดอาจเป็นประเภทข้อมูลใด ๆ ก็ได้ เช่น

```

#define TSIZE 10
#define NAMESIZE 20
#define ADDRSIZE 30
int age[TSIZE];
float size[TSIZE+1];
char name[NAMESIZE], address[ADDRSIZE];

```

จากตัวอย่างจะได้ตัวแปรชุดชื่อ age เป็นประเภท int มีสมาชิก 10 ตัว ได้ตัวแปรชุดชื่อ size เป็นประเภท float มีสมาชิก 11 ตัว ส่วนการประกาศข้อมูลสุดท้ายจะได้ตัวแปรชุดประเภท char คือ name มีสมาชิกเป็นตัวอักษร 20 ตัว และ address มีสมาชิกเป็นตัวอักษร 30 ตัว

**ตัวอย่างที่ 6.3** เขียนโปรแกรมเพื่อรับข้อมูลจำนวนจริง 3 จำนวนจากผู้ใช้และหาค่าเฉลี่ยของค่าที่รับเข้ามาทั้งหมด ให้นำว่ามีเลขจำนวนจริงที่รับเข้ามานั้นมีค่าน้อยกว่าค่าเฉลี่ยกี่จำนวน โดยใช้ตัวแปรชุดในการเก็บข้อมูล

---

```
#include <stdio.h>
#define N 3
void readData(float []);
void calAverage(float [], float *);
void findLessAverage(float [], float, int *);
void main() {
 float x[N], average;
 int num;
 readData(x);
 calAverage(x, &average);
 findLessAverage(x, average, &num);
 printf("Less than average = %d", num);
}
void readData(float px[]) {
 int i;
 for (i=0; i<N; i++) {
 printf("Enter value %d : ", i+1);
 scanf("%f", &px[i]);
 }
}
void calAverage(float a[], float *pAvg) {
 int i;
 float sum=0.0;
 for (i=0; i<N; i++) {
 sum = sum + a[i];
 }
 *pAvg = sum / N;
}
```

```

void findLessAverage(float b[], float avg, int *pNum) {
 int i;
 *pNum = 0;
 for (i=0; i<N; i++) {
 if (b[i] < avg)
 (*pNum)++;
 }
}

```

---

จากตัวอย่างที่ 6.3 จะเป็นการปรับปรุงจากตัวอย่างที่ 5.7 ซึ่งสังเกตได้ว่าตัวแปรเลขจำนวนจริง 3 ตัว คือ x1, x2, x3 ที่ใช้ในตัวอย่างที่ 5.7 นั้นมีการทำงานที่เหมือนกัน ตัวอย่างเช่นในฟังก์ชัน findLessAverage( ) จะต้องเขียนคำสั่งเปรียบเทียบเงื่อนไขที่ซ้ำกันทั้ง 3 ครั้ง การทำงานในลักษณะเช่นนี้นิยมใช้ตัวแปรชุดมาช่วย นอกจากนี้หากโจทย์เปลี่ยนใหม่เป็นให้รับข้อมูล 10 ตัว ถ้าเขียนโปรแกรมในลักษณะตัวอย่างที่ 5.7 จะต้องประกาศตัวแปร 10 ตัว และแก้ไขโปรแกรมที่อื่น ๆ แต่ในตัวอย่างที่ 6.3 สามารถแก้ไขได้ที่ตำแหน่งเดียวคือ เปลี่ยนคำสั่ง #define เป็น

```
#define N 10
```

นอกจากนี้ให้สังเกตการส่งผ่านตัวแปรชุดไปยังฟังก์ชัน เช่น การรับข้อมูลเข้ามาเก็บในตัวแปรชุดของฟังก์ชัน readData( ) สามารถกำหนดโปรโตไทป์ได้ว่า

```
void readData(float []);
```

จากโปรโตไทป์จะบอกให้ทราบว่ามีการส่งตัวแปรตัวแปรชุดชนิด float เข้าไปยังฟังก์ชัน การเรียกใช้งานจากฟังก์ชัน main( ) สามารถเรียกโดยอ้างถึงชื่อของตัวแปรชุดนั้นได้ทันที เช่น

```
float x[N];
readData(x);
```

สังเกตว่าไม่จำเป็นต้องระบุขนาดของตัวแปรชุดในฟังก์ชัน การเปลี่ยนแปลงค่าใด ๆ ของตัวแปรตัวแปรชุดในฟังก์ชัน readData( ) จะเป็นการเปลี่ยนแปลงค่าของตัวแปรชุด x ในฟังก์ชัน main( ) ด้วย จากตัวอย่างตัวแปร px ในฟังก์ชัน readData( ) จะทำหน้าที่คล้ายกับพอยน์เตอร์ชี้มายังตัวแปรชุด x ในฟังก์ชัน main( ) ซึ่งการอ้างถึงชื่อตัวแปรตัวแปรชุดจะเป็นการอ้างถึงแอดเดรสเริ่มต้นของตัวแปรชุดนั้น เพราะฉะนั้นเมื่อมีการรับข้อมูลมาเก็บยังตัวแปร px[i] ก็ะเหมือนกับการรับข้อมูลมาเก็บที่ตัวแปร x[i] นั่นเอง

แสดงตัวอย่างเพิ่มเติมดังตัวอย่างที่ 6.4 ถึง 6.6 ตัวอย่างที่ 6.4 แสดงการรับข้อมูลตัวแปรชุด 2 ตัวแปรชุดและนำมาเปรียบเทียบกัน ตัวอย่างที่ 6.5 จะแสดงตัวอย่างการใช้ตัวแปรชุดในกรณีที่ได้รับข้อมูลไม่เต็มจำนวนของสมาชิกตัวแปรชุดที่ได้จองไว้ ส่วนตัวอย่างที่ 6.6 แสดงตัวอย่างการใช้ตัวแปรชุดคู่ขนาน เป็นตัวแปรชุด 2 ตัวแปรชุดที่เก็บข้อมูลสมาชิกของทั้ง 2 ตัวแปรชุดสอดคล้องกัน

**ตัวอย่างที่ 6.4** โปรแกรมเพื่อรับข้อมูลตัวแปรชุดของจำนวนเต็ม 2 ตัวแปรชุด แต่ละตัวแปรชุดประกอบด้วยสมาชิกจำนวนเต็ม 5 จำนวน โดยมีเงื่อนไขการรับข้อมูลแต่ละตัวแปรชุด คือ สมาชิกตัวแรกเป็นเลขจำนวนเต็มใด ๆ ก็ได้ แต่ถ้าเป็นข้อมูลตัวอื่น ๆ ข้อมูลสมาชิกที่รับนั้นจะต้องมีค่ามากกว่าข้อมูลของสมาชิกก่อนหน้านั้น หลังจากรับข้อมูลแล้วให้นำข้อมูลจากตัวแปรชุดทั้ง 2 มาสร้างตัวแปรชุดผลลัพธ์ โดยเปรียบเทียบสมาชิกในตัวแปรชุดที่ 1 กับตัวแปรชุดที่ 2 ในลักษณะสมาชิกต่อสมาชิก หากสมาชิกในตัวแปรชุดใดมีค่ามากกว่าให้นำค่ามาเก็บในตัวแปรชุดที่ 3 ในตำแหน่งสมาชิกที่ตรงกัน กรณีที่มีค่าเท่ากันให้นำสมาชิกในตัวแปรชุดใดมากก็ได้ ดังตัวอย่าง

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |   |    |    |    |    |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|----|----|----|----|
| ตัวแปรชุด 1 | <table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 5px; text-align: center;">3</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">5</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">7</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">10</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">12</td> </tr> </table> | 3 | 5  | 7  | 10 | 12 |
| 3           | 5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 7 | 10 | 12 |    |    |
| ตัวแปรชุด 2 | <table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 5px; text-align: center;">2</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">5</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">9</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">10</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">11</td> </tr> </table> | 2 | 5  | 9  | 10 | 11 |
| 2           | 5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 9 | 10 | 11 |    |    |
| ตัวแปรชุด 3 | <table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 5px; text-align: center;">3</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">5</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">9</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">10</td> <td style="border: 1px solid black; padding: 5px; text-align: center;">12</td> </tr> </table> | 3 | 5  | 9  | 10 | 12 |
| 3           | 5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 9 | 10 | 12 |    |    |

---

```
#include <stdio.h>
#define SIZE 5
void readArray (int []);
void calArray (int [], int [], int []);
void printArray (int [], int [], int []);
void main() {
 int arr1[SIZE], arr2[SIZE], result[SIZE];
 printf("\nENTER FIRST DATA\n");
 readArray(arr1);

 printf("\nENTER SECOND DATA\n");
 readArray(arr2);
 printf("\n\nCOMPARE ARRAY");
 calArray(arr1, arr2, result);
 printArray(arr1, arr2, result);
}
void readArray (int a[]) {
 int i=0, tmp;
 while (i < SIZE) {
 printf("Enter member #%d : ", i);
 scanf("%d", &tmp);
 if (i==0 || tmp > a[i-1])
 a[i++] = tmp;
```

```

else
 printf("!!!ERROR!!!, data must more than %d\n", a[i-1]);
}
}

void calArray (int first[], int second[], int ans[]) {
 int i;
 for (i=0; i < SIZE; i++)
 if (first[i] > second[i])
 ans[i] = first[i];
 else
 ans[i] = second[i];
}

void printArray (int first[], int second[], int ans[]) {
 int i;

 printf("\n\nP R I N T R E S U L T");
 for (i=0; i < SIZE; i++)
 printf("\n%d. first [%d], second[%d] => ans [%d]", i, first[i], second[i], ans[i]);
}

```

---

**ตัวอย่างที่ 6.5** เขียนโปรแกรมเพื่อรับข้อมูลคะแนนสอบของนักเรียนห้องหนึ่งซึ่งมีไม่เกิน 100 คน หากมีการป้อนข้อมูลคะแนน -999 แสดงว่าสิ้นสุดการป้อนข้อมูล ให้หาว่าคะแนนเฉลี่ยของการสอบครั้งนั้นเป็นเท่าใด และคะแนนที่สูงที่สุดและต่ำที่สุดเป็นเท่าใด

---

```

#include <stdio.h>
#define MAX 100
void readScore(float [], int *);
float findAvg(float [], int);
float findMax(float [], int);
float findMin(float [], int);

void main() {
 float score[MAX], avg, max, min;
 int num;
 avg = max = min = 0.0;
 readScore(score, &num);
 if (num > 0) {
 avg = findAvg(score, num);

```

```

 max = findMax(score, num);
 min = findMin(score, num);
}
printf("\nAverage is %.2f, Max. score is %.2f, Min. score is %.2f", avg, max, min);
}
void readScore(float pscore[], int *n) {
 int i=0;
 printf("Enter student no.%d score : ", i+1);
 scanf("%f", &pscore[i]);
 while (i < MAX && pscore[i] != -999) {
 i++;
 printf("Enter student no.%d score : ", i+1);
 scanf("%f", &pscore[i]);
 }
 *n = i;
}
float findAvg(float pscore[], int n) {
 int i;
 float sum=0.0, avg;
 for (i=0; i<n; i++)
 sum = sum + pscore[i];
 avg = sum / n;
 return(avg);
}
float findMax(float pscore[], int n) {
 int i;
 float max;
 max = pscore[0];
 for (i=1; i<n; i++) {
 if (max < pscore[i])
 max = pscore[i];
 }
 return(max);
}
float findMin(float pscore[], int n) {
 int i;
 float min;
 min = pscore[0];

```

```

for (i=1; i<n; i++) {
 if (min > pscore[i])
 min = pscore[i];
}
return(min);
}

```

ในการใช้งานตัวแปรชุดของภาษาซีจะต้องมีการระบุขนาดของตัวแปรชุดที่แน่นอนในช่วงของการเขียนโปรแกรมทุกครั้ง แต่อาจจะใช้งานตัวแปรชุดไม่เต็มขนาดที่จองไว้ก็ได้ แต่ทั้งนี้ต้องคำนึงเรื่องการเสียเปล่าของหน่วยความจำที่ถูกจองพื้นที่ด้วย

**ตัวอย่างที่ 6.6** เขียนโปรแกรมเพื่อรับข้อมูลอายุและความสูงของนักเรียนห้องหนึ่งซึ่งมีนักเรียน 30 คน ให้หาความสูงเฉลี่ยของนักเรียนในห้องนั้น และแสดงรายละเอียดอายุและความสูงของนักเรียนที่สูงกว่าความสูงเฉลี่ย และหาว่าอายุเฉลี่ยของนักเรียนที่สูงกว่าความสูงเฉลี่ยเป็นเท่าใด

```

#include <stdio.h>
#define MAX 30
void readStudentData(int [], float []);
float findAverageHeight(float []);
void printHigherAverage(float, int [], float []);
void main() {
 int age[MAX];
 float height[MAX], avg;
 readStudentData(age, height);
 avg = findAverageHeight(height);
 printHigherAverage(avg, age, height);
}
void readStudentData(int pAge[], float pHeight[]) {
 int i;
 for (i=0; i<MAX; i++) {
 printf("Enter data for student no.%d\n", i+1);
 printf("Enter age : ");
 scanf("%d", &pAge[i]);
 printf("Enter height (cm.) : ");
 scanf("%f", &pHeight[i]);
 }
}

```



```

float findAverageHeight(float pHeight[]) {
 int i;
 float sum=0.0, average;
 for (i=0; i<MAX; i++) {
 sum += pHeight[i];
 }
 average = sum / MAX;
 return(average);
}

void printHigherAverage(float avgHeight, int pAge[], float pHeight[]) {
 int i, count=0;
 float sum=0.0, avgAge;
 printf("\n\nHigher than average height %.2f cm.", avgHeight);
 for (i=0; i<MAX; i++) {
 if (pHeight[i] > avgHeight) {
 printf("\nStudent no.-%d, age %d years old, height %.2f cm.", i+1, pAge[i], pHeight[i]);
 sum += pAge[i];
 count++;
 }
 }
 avgAge = sum / count;
 printf("\n\nAverage age in this group is %.2f", avgAge);
}

```

การกำหนดค่าเริ่มต้นให้กับตัวแปรชุดสามารถทำได้ในช่วงเริ่มต้นของการประกาศตัวแปร เช่น ต้องการกำหนดราคาสินค้าภายในร้านค้าแห่งหนึ่ง โดยเก็บข้อมูลเป็นตัวแปรชุด สามารถทำได้ด้วยคำสั่ง

```
float price[] = {100.0, 120.0, 85.0, 90.0, 150.0};
```

ข้อมูลภายในเครื่องหมายปีกกาจะถูกกำหนดให้เก็บในตัวแปรชุดชื่อ price ในตำแหน่งสมาชิกเริ่มต้นจาก 0 จนถึงสมาชิกตัวสุดท้าย และสังเกตว่าไม่มีการประกาศขนาดของตัวแปรชุด แต่ขนาดของตัวแปรชุดจะขึ้นอยู่กับค่าที่กำหนดให้ว่ามีจำนวนกี่ตัว ในที่นี่จะได้ว่าตัวแปรชุด price มีสมาชิก 5 ตัว แสดงได้ดังรูปที่ 6.1 และแสดงการใช้งานดังตัวอย่างที่ 6.7

|       |          |          |          |          |          |
|-------|----------|----------|----------|----------|----------|
| price | 100.0    | 120.0    | 85.0     | 90.0     | 150.0    |
|       | price[0] | price[1] | price[2] | price[3] | price[4] |

รูปที่ 6.1 แสดงข้อมูลที่เก็บในตัวแปรชุด price

**ตัวอย่างที่ 6.7** เขียนโปรแกรมเพื่อคำนวณปริมาตรน้ำมันที่ลูกค้าจะได้รับเมื่อเติมน้ำมันที่สถานีบริการ น้ำมันแห่งหนึ่ง โดยมีประเภทน้ำมันอยู่ 3 ประเภทและเก็บราคาอยู่ในตัวแปรชุดดังนี้

สมาชิกที่ 0 เก็บราคาน้ำมันเบนซิน 91 มีค่า 14.50 บาท

สมาชิกที่ 1 เก็บราคาน้ำมันเบนซิน 95 มีค่า 15.50 บาท

สมาชิกที่ 2 เก็บราคาน้ำมันดีเซล มีค่า 12.50 บาท

---

```
#include <stdio.h>

#define MAX 3

void readData(int *, float *, float []);
float calVolume(int, float, float []);
void main() {
 int type;
 float volume, amount, price[]={14.5, 15.5, 12.5};
 readData(&type, &amount, price);
 volume = calVolume(type, amount, price);
 printf("\n\t\t\tYou will got %.2f litres", volume);
}

void readData(int *t, float *a, float price[]) {
 int i;
 printf("\n\t\t\tFuel price");
 for (i=0; i<MAX; i++) {
 printf("\n\t\t%d. ", i+1);
 switch (i) {
 case 0 : printf("%-15s -> ", "Benzene 91");
 break;
 case 1 : printf("%-15s -> ", "Benzene 95");
 break;
 default : printf("%-15s -> ", "Diesel");
 }
 printf("%.2f Baht/litre", price[i]);
 }
 do {
 printf("\n\t\t\tSelect type (1-3) : ");
 scanf("%d", t);
 } while (!(*t >= 1 && *t <= 3));
 do {
 printf("\n\t\t\tAmount (Baht) : ");
 scanf("%f", a);
 }
```

```

 } while (!(*a > 0));
}
float calVolume(int type, float amount, float price[]) {
 int vol;
 vol = amount / price[type-1];
 return(vol);
}

```

## 2. การใช้พอยน์เตอร์กับตัวแปรชุด

การทำงานใด ๆ ของตัวแปรชุดสามารถใช้พอยน์เตอร์เข้ามาช่วย ซึ่งจะช่วยให้มีความเร็วในการทำงานสูงขึ้น สมมติว่ามีตัวแปรชุด  $a$  และพอยน์เตอร์  $pa$  ดังนี้

```

int a[10];
int *pa;

```

เนื่องจากการอ้างถึงชื่อของตัวแปรชุดจะเป็นการอ้างถึงแอดเดรสเริ่มต้นของตัวแปรชุดนั้น หากต้องการให้ตัวแปรพอยน์เตอร์  $pa$  มาชี้ยังตัวแปรชุด  $a$  สามารถทำได้โดยใช้คำสั่ง

```
pa = a;
```

ซึ่งมีการทำงานจะเหมือนกับการใช้คำสั่ง

```
pa = &a[0];
```

$pa$  จะเก็บค่าแอดเดรสเริ่มต้นของตัวแปรชุด  $a$  แสดงดังรูปที่ 6.2



รูปที่ 6.2 แสดงพอยน์เตอร์ชี้ไปยังแอดเดรสเริ่มต้นของตัวแปรชุด

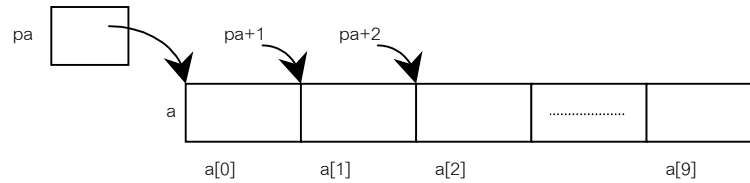
การนำพอยน์เตอร์นั้นไปใช้งานสามารถทำได้โดยอ่านค่าตัวแปรชุดผ่านพอยน์เตอร์ดังนี้

```
int x = *pa;
```

จะเป็นการกำหนดค่าให้  $x$  มีค่าเท่ากับ  $a[0]$  การเลื่อนไปอ่านค่าสมาชิกตำแหน่งต่าง ๆ ของตัวแปรชุดผ่านทางพอยน์เตอร์สามารถทำได้โดยการเพิ่มค่าพอยน์เตอร์ขึ้น 1 เพื่อเลื่อนไปยังตำแหน่งถัดไป หรือเพิ่มค่าขึ้น  $N$  เพื่อเลื่อนไป  $N$  ตำแหน่ง หรืออาจจะลดค่าเพื่อเลื่อนตำแหน่งลง กรณีที่  $pa$  ชี้อยู่ที่  $a[0]$  คำสั่ง

pa+1;

จะเป็นการอ้างถึงแอดเดรสของ a[1] หากเป็น pa+i เป็นการอ้างถึงแอดเดรส a[i] หากต้องการอ้างถึงข้อมูลภายในของสมาชิกของตัวแปรชุดตำแหน่งที่ a[i] จะใช้ \*(pa+i) แสดงดังรูปที่ 6.3



รูปที่ 6.3 แสดงการอ้างถึงตำแหน่งในตัวแปรชุดผ่านพอยน์เตอร์

การสั่งให้บวก 1 หรือบวก i หรือ ลบ i เป็นเหมือนการเลื่อนไปยังสมาชิกของตัวแปรชุดตำแหน่งที่ต้องการ เนื่องจากประเภทของข้อมูลแต่ละประเภทของตัวแปรชุด เช่น int, float, float และอื่น ๆ มีขนาดของข้อมูลที่แตกต่างกัน ทำให้ขนาดของสมาชิกภายในตัวแปรชุดแต่ละประเภทมีขนาดแตกต่างกันด้วย การสั่งให้บวกหรือลบด้วยจำนวนที่ต้องการนั้นจะมีกลไกที่ทำหน้าที่คำนวณตำแหน่งที่ต้องการให้สอดคล้องกับข้อมูลแต่ละประเภทโดยอัตโนมัติ

นอกจากนี้ยังสามารถใช้พอยน์เตอร์แทนตัวแปรชุด การอ้างโดยใช้ a[i] สามารถใช้ \*(a+i) เนื่องจากทุกครั้งที่อ้างถึง a[i] ภาษาซีจะทำหน้าที่แปลงเป็น \*(a+i) เพราะฉะนั้นการเขียนในรูปแบบใดก็ให้ผลลัพธ์ในการทำงานเช่นเดียวกัน และการอ้างถึงแอดเดรส เช่น &a[i] จะมีผลเท่ากับการใช้ a+i

ในลักษณะเดียวกันการใช้งานพอยน์เตอร์ก็สามารถใช้คำสั่งในลักษณะตัวแปรชุดก็ได้ เช่น การอ้างถึง \*(pa+i) สามารถเขียนด้วย pa[i] ก็ได้ผลเช่นเดียวกัน

สิ่งที่แตกต่างกันของตัวแปรชุดและพอยน์เตอร์ คือ พอยน์เตอร์เป็นตัวแปร แต่ตัวแปรชุดไม่ใช่ตัวแปร สมมติให้ a เป็นตัวแปรชุดและ pa เป็นพอยน์เตอร์ การอ้างถึง pa = a หรือ pa++ จะสามารถคอมไพล์ได้ แต่จะไม่สามารถใช้คำสั่ง a = pa หรือ a++ ได้

เมื่อมีการส่งชื่อของตัวแปรชุดให้แก่ฟังก์ชัน จะเป็นการส่งตำแหน่งแอดเดรสของสมาชิกตัวแรกของตัวแปรชุดให้แก่ฟังก์ชัน ดังนั้นพารามิเตอร์ในฟังก์ชันนั้นจะเป็นตัวแปรประเภทพอยน์เตอร์ แสดงดังตัวอย่างที่ 6.8

**ตัวอย่างที่ 6.8** เป็นการแปลงโปรแกรมในตัวอย่าง 6.6 ให้เป็นการเขียนในลักษณะพอยน์เตอร์ทั้งหมด ซึ่งให้ผลลัพธ์การทำงานที่เหมือนกัน

```
#include <stdio.h>
#define MAX 5
void readStudentData(int *, float *);
float findAverageHeight(float *);
void printHigherAverage(float, int *, float *);
```

```

void main() {
 int age[MAX];
 float height[MAX], avg;
 readStudentData(age, height);
 avg = findAverageHeight(height);
 printHigherAverage(avg, age, height);
}

void readStudentData(int *pAge, float *pHeight) {
 int i;
 for (i=0; i<MAX; i++) {
 printf("Enter data for student no.%d\n", i+1);
 printf("Enter age : ");
 scanf("%d", pAge+i);
 printf("Enter height (cm.) : ");
 scanf("%f", pHeight+i);
 }
}

float findAverageHeight(float *pHeight) {
 int i;
 float sum=0.0, average;
 for (i=0; i<MAX; i++) {
 sum += *(pHeight+i);
 }
 average = sum / MAX;
 return(average);
}

void printHigherAverage(float avgHeight, int *pAge, float *pHeight) {
 int i, count=0;
 float sum=0.0, avgAge;
 printf("\n\nHigher than average height %.2f cm.", avgHeight);
 for (i=0; i<MAX; i++) {
 if (*(pHeight+i) > avgHeight) {
 printf("\nStudent no.-%2d, age %d years old, height %.2f cm.",
 i+1, *(pAge+i), *(pHeight+i));
 sum += *(pAge+i);
 count++;
 }
 }
}

```

```

avgAge = sum / count;
printf("\n\nAverage age in this group is %.2f", avgAge);
}

```

### 3. ตัวแปรชุดของตัวอักษร

ในกรณีของการใช้ตัวแปรชุดกับข้อมูลชนิด char จะมองเป็นการทำงานกับข้อมูลหลาย ๆ ตัวอักษรหรือข้อความที่เรียกว่า สตริง (String) ค่าคงที่ของข้อความสิ่งที่มีการใช้เสมอในโปรแกรม เช่น "Hello" แต่ในการใช้งานตัวแปรสตริงจะต้องมีการเตรียมพื้นที่ในการเก็บข้อความเอาไว้หนึ่งตำแหน่งเสมอ เช่น หากต้องการประกาศตัวแปรเพื่อเก็บข้อความว่า "Mickey Mouse" จะต้องจองพื้นที่เท่ากับจำนวนตัวอักษรที่มีและบวกไปด้วย 1 เสมอ เนื่องจากลักษณะการเก็บข้อมูลประเภทข้อความในหน่วยความจำจะมีการปะตัวอักษร Null หรือ '\0' ต่อท้ายเสมอเพื่อให้รู้ว่าเป็นจุดสิ้นสุดของข้อมูล ในที่นี้ต้องจองพื้นที่ขนาด 13 ตัวอักษร การจองพื้นที่ดังกล่าวจะเหมือนการจองพื้นที่ของข้อมูลประเภทตัวแปรชุดเป็นตัวแปรชุดของ char สามารถใช้คำสั่งในการประกาศตัวแปรคือ

```
char message[13];
```

หากต้องการกำหนดค่าให้กับตัวแปร message สามารถทำได้ด้วยคำสั่ง

```
char message[] = "Mickey";
```

จำลองการเก็บข้อมูลดังกล่าวในหน่วยความจำดังรูปที่ 6.4

|   |   |   |   |   |   |  |   |   |   |   |   |    |
|---|---|---|---|---|---|--|---|---|---|---|---|----|
| M | i | c | k | e | y |  | M | o | u | s | e | \0 |
|---|---|---|---|---|---|--|---|---|---|---|---|----|

รูปที่ 6.4 แสดงแบบจำลองการเก็บข้อมูลประเภทสตริงในหน่วยความจำ

ค่าคงที่สตริงที่พบเห็นได้เสมอได้แก่ข้อความที่ใช้ในฟังก์ชัน printf( ) เช่น

```
printf ("Hello, world\n");
```

ฟังก์ชัน printf( ) จะรับพารามิเตอร์เป็นพอยน์เตอร์ชี้ไปยังแอดเดรสของข้อมูลที่ตำแหน่งเริ่มต้นของตัวแปรชุด และนำข้อความนั้นแสดงออกทางอุปกรณ์แสดงข้อมูลมาตรฐาน ในการเขียนโปรแกรมจะสามารถใช้พอยน์เตอร์ชี้ไปค่าคงที่สตริงใด ๆ ก็ได้ เช่น

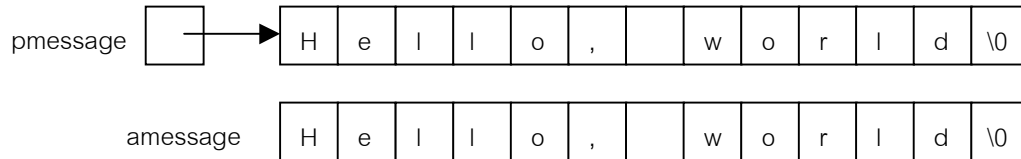
```
char *pmessage = "Hello, world";
```

pmessage จะเป็นพอยน์เตอร์ประเภท char ชี้ไปที่ตัวแปรชุดของตัวอักษร จะแตกต่างจากการใช้ตัวแปรชุดทั่วไปเช่น

```
char amessage[] = "Hello, world";
```

ลักษณะของตัวแปรชุดเช่น amessage จะมีการจองพื้นที่ให้กับตัวแปรชุดขนาด 13 ตัวอักษรรวมทั้ง Null ส่วนลักษณะของพอยน์เตอร์ที่ชี้ไปยังค่าคงที่สตริง จะมีการจองพื้นที่ให้กับค่าคงที่สตริงขนาด 13 ตัวอักษรเช่นเดียว

กัน แต่จะมีการจองพื้นที่ให้กับพอยน์เตอร์และทำการชี้พอยน์เตอร์นั้นไปยังพื้นที่ของค่าคงที่สตริงที่จองเอาไว้ แสดงดังรูปที่ 6.5 และแสดงตัวอย่างการใช้งานดังตัวอย่างที่ 6.9



รูปที่ 6.5 แสดงการจองพื้นที่ให้กับตัวแปรชุดและพอยน์เตอร์ชี้ไปยังค่าคงที่สตริง

**ตัวอย่างที่ 6.9** ฟังก์ชันที่รับพารามิเตอร์เป็นพอยน์เตอร์ โดยอาร์กิวเมนต์ที่ส่งมาเป็นตัวแปรชุด เป็นการหาความยาวข้อความที่ส่งเข้ามายังฟังก์ชัน

```
#include <stdio.h>
int strlen(char *s) {
 int n;
 for (n=0; *s!='\0'; s++)
 n++;
 return n;
}
void main() {
 char str[]="I love C";
 int len;
 len = strlen(str);
 printf("Length of %s is %d", str, len);
}
```

จะเห็นว่า s เป็นพอยน์เตอร์ ในฟังก์ชันจะมีการตรวจสอบข้อมูลว่ามีค่าเท่ากับ '\0' หรือไม่ และมีการเลื่อนตำแหน่งทีละ 1 ค่า (นับว่าข้อมูลมีความยาวเพิ่มขึ้นทีละ 1) โดยใช้ s++ ค่าตอบที่ได้จากการเรียกใช้ฟังก์ชันนี้จะเป็นความยาวของข้อมูลที่เก็บอยู่ภายในโดยที่ไม่รวมค่า Null การเรียกใช้ฟังก์ชัน strlen สามารถทำได้หลายลักษณะ

```
strlen ("Hello world"); /* string constant */
strlen (array); /* char array[]="Hello world"; */
strlen (ptr); /* char *ptr="Hello world"; */
```

นอกจากนี้ยังอาจจะประกาศพารามิเตอร์ภายในฟังก์ชัน strlen ได้ใน 2 ลักษณะ คือ char \*s แบบในตัวอย่าง หรืออาจจะใช้ char s[] ก็ได้ โดยทั่วไปจะใช้ในลักษณะแรก เพราะช่วยในรู้ได้ทันทีว่า s เป็นตัวแปรพอยน์

เตอร์ และยังสามารถส่งส่วนใดส่วนของตัวแปรชุดให้แก่ฟังก์ชันก็ได้ โดยไม่จำเป็นต้องส่งสมาชิกตัวแรกก็ได้เช่นกัน  
เช่น

```
strlen(&str[2]) หรือ strlen(str+2)
```

เป็นการส่งแอดเดรสของสมาชิก str[2] ให้กับฟังก์ชัน strlen() การประกาศฟังก์ชัน strlen() สามารถทำได้โดยการประกาศ

```
int strlen(char s[]) { } หรือ int strlen(char *arr) { }
```

แสดงตัวอย่างเพิ่มเติมของการทำงานของสตริงดังตัวอย่างที่ 6.10 ถึง 6.12 เป็นการใช้ฟังก์ชัน strcpy() ที่ทำหน้าที่ในการทำสำเนา (Copy) ข้อมูลจากตัวแปรหนึ่งไปยังอีกตัวแปรหนึ่ง สามารถเขียนโดยในพารามิเตอร์ในลักษณะตัวแปรชุดและพอยน์เตอร์ได้ดังตัวอย่าง

**ตัวอย่างที่ 6.10** ฟังก์ชัน strcpy () ทำหน้าที่สำเนาข้อมูลจากตัวแปรหนึ่งไปยังอีกตัวแปรหนึ่งเขียนในลักษณะตัวแปรชุด

---

```
void strcpy (char *s, char *t) {
 int i=0;
 while ((s[i] = t[i]) != '\0')
 i++;
}
```

---

**ตัวอย่างที่ 6.11** ฟังก์ชัน strcpy () เขียนในลักษณะพอยน์เตอร์

---

```
void strcpy (char *s, char *t) {
 while ((*s = *t) != '\0') {
 s++;
 t++;
 }
}
```

---

ฟังก์ชันทั้ง 2 ลักษณะจะทำงานเหมือนกัน t และ s จะชี้ไปที่ตัวแปรชุดของ char และมีการทำสำเนาค่าข้อมูลที่ของ s ชี้ไปที่ให้กับ t ทีละ 1 ตัวอักษร จนกว่าค่าที่สำเนานั้นจะเป็น Null แต่จะสามารถเขียนฟังก์ชันดังกล่าวสั้น ๆ ได้ดังนี้



### ตัวอย่างที่ 6.12 ฟังก์ชัน strcpy ( ) เขียนในลักษณะพอยน์เตอร์แบบสั้น

```
void strcpy (char *s, char *t){
 while ((*s++ = *t++) != '\0');
}
```

#### 4. การคำนวณกับแอดเดรส

ให้ p เป็นพอยน์เตอร์ชี้ไปยังตัวแปรชุดใด ๆ คำสั่ง p++ เป็นการเลื่อน p ไปยังสมาชิกถัดไป และคำสั่ง p += i เป็นการเลื่อนพอยน์เตอร์ไป i ตำแหน่งจากตำแหน่งปัจจุบัน นอกจากนี้ยังสามารถใช้เครื่องหมายความสัมพันธ์ (Relational Operator) เช่น ==, !=, <, >= และอื่น ๆ ทำงานร่วมกับพอยน์เตอร์ได้ สมมติให้ p และ q ชี้ไปยังสมาชิกของตัวแปรชุดเดียวกัน เช่น

```
char msg[] = "Hello";
char *p, *q;
p = msg;
q = msg+2;
if (p < q)
```

จะเป็นจริงเมื่อ p ชี้ไปที่สมาชิกที่อยู่ก่อนหน้าสมาชิกที่ q ชี้อยู่ การเปรียบเทียบในลักษณะจะใช้ได้ต่อเมื่อ p และ q ชี้ไปที่ตัวแปรชุดเดียวกันเท่านั้น

นอกจากนี้ยังสามารถใช้การลบหรือการบวกกับพอยน์เตอร์ได้เช่นเดียวกัน แต่สิ่งที่ควรระวังคือ การทำเช่นนั้นจะต้องอยู่ในขอบเขตขนาดของตัวแปรชุดเท่านั้น สามารถปรับปรุงฟังก์ชัน strlen ( ) ใหม่ให้ทำงานกระชับขึ้นดังตัวอย่างที่ 6.13

### ตัวอย่างที่ 6.13 ฟังก์ชัน strlen ( ) ปรับปรุงให้กระชับขึ้น

```
/* strlen : return length of string s */
int strlen (char *s) {
 char *p = s;
 while (*p != '\0')
 p++;
 return p-s;
}
```

เนื่องจาก s ซึ่อยู่ที่ตำแหน่งเริ่มต้น โดยมี p ซึ่ไปที่ s เช่นเดียวกัน แต่จะมีการเลื่อน p ไปทีละหนึ่งตำแหน่ง จนกว่าค่าที่ตำแหน่งที่ p ซึ่อยู่จะเท่ากับ '\0' เมื่อนำ p ค่าสุดท้ายมาลบกับ s ที่ตำแหน่งเริ่มต้นก็จะได้ความยาวของข้อมูลที่ส่งเข้ามา แสดงตัวอย่างเพิ่มเติมเกี่ยวกับการใช้ข้อมูลสตริงดังตัวอย่างที่ 6.14

**ตัวอย่างที่ 6.14** ให้เขียนโปรแกรมเพื่อรับข้อความหนึ่งจากผู้ใช้ ให้ตรวจสอบว่าข้อความดังกล่าวมีตัวอักษรที่เป็นสระ A E I O U อยู่ทั้งหมดกี่ตัวอักษร

---

```
#include <stdio.h>
#define MAX 50
int countVowel(char []);
void main() {
 char text[MAX];
 int cVowel;

 printf("Enter text : ");
 scanf("%s", text);

 cVowel = countVowel(text);

 printf("Text : [%s] has %d vowels", text, cVowel);
}

int countVowel(char t[]) {
 int i=0, count=0;
 while (i<MAX && t[i]!='\0') {
 if (t[i]=='A' || t[i]=='a' || t[i]=='E' || t[i]=='e' || t[i]=='I' || t[i]=='i' ||
 t[i]=='O' || t[i]=='o' || t[i]=='U' || t[i]=='u')
 count++;
 i++;
 }
 return(count);
}
```

---

จากตัวอย่างจะเห็นว่าจะมีการเปรียบเทียบเงื่อนไขว่าข้อมูลนั้นเป็นตัวอักษรตัวพิมพ์เล็กหรือตัวพิมพ์ใหญ่ ซึ่งมีฟังก์ชันมาตรฐานที่ช่วยอำนวยความสะดวกดังกล่าว คือ ฟังก์ชัน toupper( ) เป็นฟังก์ชันในการแปลงตัวอักษรจากตัวอักษรใด ๆ ไปเป็นตัวอักษรตัวพิมพ์ใหญ่ ซึ่งก่อนจะใช้งานต้องมีการเรียกใช้อินคลูซไฟล์ ctype.h ปรับปรุงได้ ตัวอย่างที่ 6.14 โดยมีการใช้ฟังก์ชัน toupper( ) และเปลี่ยนตัวแปรในฟังก์ชัน countVowel( ) เป็นแบบพอยน์เตอร์ได้ ดังตัวอย่างที่ 6.15

**ตัวอย่างที่ 6.15** ให้เขียนโปรแกรมเพื่อรับข้อความหนึ่งจากผู้ใช้ ให้ตรวจสอบว่าข้อความดังกล่าวมีตัวอักษรที่เป็นสระ A E I O U อยู่ทั้งหมดกี่ตัวอักษร โดยใช้พอยน์เตอร์

---

```
#include <stdio.h>
#include <ctype.h>
#define MAX 50
int countVowel(char *);
void main() {
 char text[MAX];
 int cVowel;
 printf("Enter text : ");
 scanf("%s", text);
 cVowel = countVowel(text);
 printf("Text : [%s] has %d vowels", text, cVowel);
}
int countVowel(char *t) {
 int count=0;
 while (*t!='\0') {
 if (toupper(*t)=='A' || toupper(*t)=='E' || toupper(*t)=='I' || toupper(*t)=='O' ||
 toupper(*t)=='U')
 count++;
 t++;
 }
 return(count);
}
```

---

## 5. ฟังก์ชันมาตรฐานของสตริง

ภาษาซีได้เตรียมฟังก์ชันมาตรฐานของการทำงานที่เกี่ยวข้องกับสตริงซึ่งพบในคอมไพเลอร์ของภาษาซีทั่วไป ก่อนจะใช้งานเหล่านี้จะต้องเรียกอินคลูซไฟล์ string.h ด้วยคำสั่ง #include <string.h>

ฟังก์ชันที่มีการนำมาใช้งานบ่อย ๆ ได้แก่

| ชื่อฟังก์ชัน                                          | หน้าที่                                                                                                                                                   |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| char *strcat(char *s1, const char *s2)                | นำสตริง s2 ไปต่อท้ายสตริง s1                                                                                                                              |
| char *strchr(const char *s, int c)                    | หาว่ามีค่าในตัวแปร c ปรากฏในตัวแปร s ตำแหน่งแรกที่ได้ จะคืนค่าพอยน์เตอร์ของตำแหน่งแรกที่พบ                                                                |
| int strcmp(const char *s1, const char *s2)            | เปรียบเทียบค่าของ s1 และ s2 ทีละตัวอักษร ตามค่าของรหัส ASCII หากมีค่าเท่ากันคืนค่าบวก หาก s1 มีค่ามากกว่า s2 คืนค่า 1 และคืนค่าลบ หาก s1 มีค่าน้อยกว่า s2 |
| char *strcpy(char *s1, const char *s2)                | สำเนาค่าจาก s2 มาใส่ใน s1                                                                                                                                 |
| Size_t strlen(const char *s)                          | หาความยาวของข้อมูลในเก็บอยู่ใน s โดยไม่นับค่า Null                                                                                                        |
| char *strncat(char *s1, const char *s2, size_t n)     | นำค่าที่อยู่ใน s2 เป็นจำนวน n ตัวอักษรไปต่อท้ายของ s1                                                                                                     |
| int strncmp(const char *s1, const char *s2, size_t n) | เปรียบเทียบค่าที่อยู่ใน s1 และ s2 เป็นจำนวน n ตัวอักษร                                                                                                    |
| char *strncpy(char *s1, const char *s2, size_t n)     | สำเนาค่าจาก s2 มาใส่ใน s1 เป็นจำนวน n ตัวอักษร โดยแทนที่เฉพาะ n ตัวอักษรแรกของ s1 เท่านั้น                                                                |
| char *strrchr(const char *s, int c)                   | หาว่ามีค่าในตัวแปร c ปรากฏในตัวแปร s ตำแหน่งแรกที่ได้ จะคืนค่าพอยน์เตอร์ของตำแหน่งสุดท้ายที่พบ                                                            |

แสดงตัวอย่างการทำงานของฟังก์ชันต่าง ๆ ของสตริงดังตัวอย่างที่ 6.16

**ตัวอย่างที่ 6.16** โปรแกรมตัวอย่างการใช้งานฟังก์ชันสตริงต่าง ๆ ที่พบบ่อย

```
#include <stdio.h>
#include <string.h>
void main() {
 char txt1[20], txt2[20], *txt3;
 int len;

 strcpy(txt1, "Hello ");
 strcpy(txt2, "world");
 strcat(txt1, txt2);
 printf("After strcat() txt1 : [%s]", txt1); /* After strcat() txt1 : [Hello world] */
 txt3 = strchr(txt1, 'w');
 printf("\n\nAfter strchr() txt3 : [%s]", txt3); /* After strchr() txt3 : [world] */
}
```

```

printf("\n\nCompare string"); /* Compare string */
strcpy(txt1, "Somchai");
strcpy(txt2, "Somsri");
if (strcmp(txt1, txt2) == 0)
 printf("\n%s equals to %s", txt1, txt2);
else if (strcmp(txt1, txt2) > 0)
 printf("\n%s greater than %s", txt1, txt2);
else
 printf("\n%s less than %s", txt1, txt2); /* Somchai less than Somsri */
len = strlen(txt1);
printf("\n\n[%s] has %d characters", txt1, len); /* [Somchai] has 7 characters */

printf("\n\nCompare first three characters"); /* Compare first three characters */
if (strncmp(txt1, txt2, 3) == 0)
 printf("\n%s equals to %s", txt1, txt2); /* Somchai equals to Somsri */
else if (strncmp(txt1, txt2, 3) > 0)
 printf("\n%s greater than %s", txt1, txt2);
else
 printf("\n%s less than %s", txt1, txt2);

strncat(txt1, txt2, 3);
printf("\n\nAfter strncat() txt1 : [%s]", txt1); /* After strncat() txt1 : [SomchaiSom] */

strncpy(txt1, "Saijai", 3);
printf("\n\nAfter strncpy() txt1 : [%s]", txt1); /* After strncpy() txt1 : [SaichaiSom] */

txt3 = strrchr(txt1, 'i');
printf("\n\nAfter strrchr() txt3 : [%s]", txt3); /* After strrchr() txt3 : [iSom] */
}

```

---



---

## 6. ตัวแปรชุดแบบหลายมิติ (Multi-dimensional Arrays)

จากพื้นฐานที่ผ่านมาเรื่องตัวแปรชุดจะเป็นลักษณะของตัวแปรชุดมิติเดียว แต่ตัวแปรชุดอาจจะมีมากกว่า 1 มิติก็ได้ ข้อมูลตัวแปรชุด 2 มิติเป็นข้อมูลที่สามารถพบเห็นได้บ่อยในชีวิตประจำวัน เช่น ตารางค่าโดยสารรถไฟ จากเมืองหนึ่งไปยังอีกเมืองหนึ่ง ตารางระยะทางจากเมืองหนึ่งไปยังอีกเมืองหนึ่ง ตารางสูตรคูณ เป็นต้น

จะสังเกตว่าข้อมูลในลักษณะของตัวแปรชุด 2 มิติจะเป็นในลักษณะของตาราง พิจารณาจากตัวอย่างการเก็บข้อมูลคะแนนสอบของนักศึกษาในกระบวนวิชาหนึ่ง ซึ่งแบ่งการเก็บคะแนนออกเป็น 5 ครั้ง จะพบว่าหากต้องการเก็บข้อมูลคะแนนสอบของนักศึกษาแต่ละคนสามารถใช้ตัวแปรชุดมิติเดียว แสดงดังรูปที่ 6.6 และใช้คำสั่ง

```
#define NUMBER_OF_PAPERS 5
float score[NUMBER_OF_PAPERS];
```

| score[0] | score[1] | score[2] | score[3] | score[4] |
|----------|----------|----------|----------|----------|
| 5.6      | 8.5      | 12.6     | 24.1     | 16.0     |

รูปที่ 6.6 แสดงการเก็บคะแนนสอบของนักศึกษาคนหนึ่ง

แต่หากเพิ่มเติมว่าให้เก็บข้อมูลของนักศึกษาทุกคนในชั้นนั้น จะต้องใช้ตัวแปรชุดหลายมิติเข้ามาเกี่ยวข้อง ตัวอย่างของข้อมูลที่เก็บดังรูปที่ 6.7

|       | ครั้งที่ 1 | ครั้งที่ 2 | ครั้งที่ 3 | ครั้งที่ 4 | ครั้งที่ 5 |
|-------|------------|------------|------------|------------|------------|
| นาย ก | 5.6        | 8.5        | 12.6       | 24.1       | 16.0       |
| นาย ข | 6.0        | 7.2        | 15.0       | 25.0       | 18.0       |
| ..... | .....      | .....      | .....      | .....      | .....      |

รูปที่ 6.7 แสดงตัวอย่างการเก็บข้อมูลคะแนนของนักศึกษา

ข้อมูลที่จะจัดเก็บในตัวแปรชุดจะต้องเป็นข้อมูลชนิดเดียวกันเสมอ ในที่นี้จะจัดเก็บเฉพาะส่วนของคะแนนสอบของนักศึกษาเท่านั้น จากลักษณะความต้องการเก็บข้อมูลดังกล่าวจะต้องเตรียมตัวแปรชุดเพื่อเก็บข้อมูลในลักษณะ 2 มิติ สามารถประกาศตัวแปรชุดดังนี้

```
#define NUMBER_OF_PAPERS 5
#define NUMBER_OF_STUDENTS 50
float score[NUMBER_OF_STUDENTS][NUMBER_OF_PAPERS];
```

ตัวแปรชุด 2 มิติจะมองข้อมูลในลักษณะแถวและคอลัมน์ แถวของข้อมูลในที่นี้จะเป็นคะแนนที่นักศึกษาแต่ละคนได้รับ ส่วนคอลัมน์จะเป็นคะแนนสอบแต่ละครั้งของนักศึกษา เพราะฉะนั้นเมื่อเราอ้างอิงจุดใดจุดหนึ่งในตัวแปรชุด 2 มิติ ก็จะเป็นคะแนนที่นักศึกษาแต่ละคนได้รับในการสอบครั้งที่ระบุ เช่น อ้างถึงแถวที่ 0 คอลัมน์ที่ 4 จะเป็นคะแนนสอบครั้งที่ 5 ของนักศึกษาคนที่ 1 การอ้างอิงข้อมูลในตัวแปรชุด 2 มิติใช้วิธีดังนี้ ดังนี้

```
score[row][column] เช่น score[0][4]
```

การทำงานของตัวแปรชุด 2 มิติจะเหมือนกับตัวแปรชุดมิติเดียว โดยจะดูว่าเป็นตัวแปรชุดที่มีสมาชิกเป็นตัวแปรชุด 1 มิติ สามารถแสดงรูปจำลองของตัวแปรชุด 2 มิติดังรูปที่ 6.8

|     |   | คอลัมน์ |     |     |     |     |
|-----|---|---------|-----|-----|-----|-----|
|     |   | 0       | 1   | 2   | 3   | 4   |
| แถว | 0 | 0,0     | 0,1 | 0,2 | 0,3 | 0,4 |
|     | 1 | 1,0     | 1,1 | 1,2 | 1,3 | 1,4 |
|     | 2 | 2,0     | 2,1 | 2,2 | 2,3 | 2,4 |
|     | ⋮ |         |     |     |     |     |

← การอ้างอิงในลักษณะแถวและคอลัมน์ เช่น score[0][4]

รูปที่ 6.8 แสดงแบบจำลองของตัวแปรชุด 2 มิติ

การจัดเก็บตัวแปรชุด 2 มิติในหน่วยความจำ เนื่องจากการเก็บข้อมูลในหน่วยความจำจะเก็บเรียงตามลำดับ ภาษาโปรแกรมแต่ละภาษาจะมีการจัดเก็บตัวแปรชุดหลายมิติในหน่วยความจำแตกต่างกัน โดยผ่านทาง Storage Mapping Function ตัวอย่างการจัดเก็บตัวแปรชุด 2 มิติ

```
int table[3][4];
```

จะมีการจัดเก็บข้อมูลเรียงลำดับภายในหน่วยความจำดังรูปที่ 6.9 เมื่อต้องการเข้าถึงสมาชิกตัวแปรชุด table[1][3] จะตรงกับตำแหน่งที่ 7 ในหน่วยความจำ (สมมติให้เริ่มนับจาก 0) ฟังก์ชันที่ใช้ในการหาคือ table[i][j] จะตรงกับตำแหน่ง  $4 * i + j$  ฟังก์ชันที่ใช้หาตำแหน่งของตัวแปรชุดหลายมิติในหน่วยความจำจะขึ้นอยู่กับค่าที่อยู่ในสูตร ในกรณีของตัวแปรชุด 2 มิติค่าที่ใช้คือ 4 หากเป็นตัวแปรชุดหลายมิติที่ไม่ใช่ 2 จะต้องหาค่าคงที่ที่จะเป็นตัวกำหนดนี้ ทั้งนี้คอมไพเลอร์จะทำหน้าที่ในการหาตำแหน่งในหน่วยความจำให้กับผู้เขียนโปรแกรมโดยอัตโนมัติ

|             |   |
|-------------|---|
| table[0][0] | 0 |
| table[0][1] | 1 |
| table[0][2] | 2 |
| table[0][3] | 3 |
| table[1][0] | 4 |
| table[1][1] | 5 |
| table[1][2] | 6 |
| table[1][3] | 7 |

รูปที่ 6.9 แสดงตัวอย่างการจัดเก็บข้อมูลตัวแปรชุด 2 มิติในหน่วยความจำ

## 7. การกำหนดค่าเริ่มต้นให้ตัวแปรชุด 2 มิติ

เนื่องจากเมื่อมีการประกาศตัวแปรชุดจะมีการจองพื้นที่ให้ตัวแปรชุด โดยที่ค่าที่อยู่ในพื้นที่หน่วยความจำที่จองอาจจะมีค่าใด ๆ ที่เราไม่ต้องการอยู่ การกำหนดค่าเริ่มต้นให้กับตัวแปรชุดเป็นการทำงานเพื่อให้สมาชิกของตัวแปรชุดมีค่าเริ่มต้นที่เราต้องการ สามารถกำหนดค่าเริ่มต้นได้ดังนี้

```
int number[][5]={{1, 2, 3, 4, 5}, {2, 4, 6, 8, 10}, {1, 3, 5, 7, 9}};
```

การกำหนดค่าเริ่มต้นจะทำพร้อมกับการประกาศตัวแปรชุดเท่านั้น โดยไม่จำเป็นต้องกำหนดขนาดจำนวนแถวของตัวแปรชุดก็ได้ เนื่องจากเมื่อมีการกำหนดค่าเริ่มต้นจะมีการจองพื้นที่ตัวแปรชุดให้เท่ากับข้อมูลที่ใช้ในการกำหนดค่าเริ่มต้นนั้น ในที่นี้จะจองตัวแปรชุด 2 มิติขนาด 3 แถว 5 คอลัมน์ เครื่องหมาย { } จะเป็นส่วนของการกำหนดค่าในแต่ละแถว ในที่นี้จะสามารถจำลองการเก็บข้อมูลได้ดังรูปที่ 6.10

| number | [0] | [1] | [2] | [3] | [4] |
|--------|-----|-----|-----|-----|-----|
| [0]    | 1   | 2   | 3   | 4   | 5   |
| [1]    | 2   | 4   | 6   | 8   | 10  |
| [2]    | 1   | 3   | 5   | 7   | 9   |

รูปที่ 6.10 แสดงการเก็บข้อมูลในตัวแปร number

## 8. การใช้งานตัวแปรชุด 2 มิติ

ดังที่กล่าวมาแล้วว่าการส่งผ่านตัวแปรชุดมิติเดียวไปยังฟังก์ชันเป็นการทำงานในลักษณะเดียวกับพอยน์เตอร์ การใช้งานตัวแปรชุด 2 มิติหรือหลายมิติกับฟังก์ชันก็เป็นการทำงานในลักษณะเดียวกัน การเปลี่ยนแปลงค่าใด ๆ ในฟังก์ชันจะมีผลกระทบต่อตัวแปรตัวแปรชุดที่ส่งเข้าไปยังฟังก์ชันนั้นด้วย การทำงานภายในตัวแปรชุด 2 มิติผู้เขียนต้องสังเกตความสัมพันธ์ระหว่างสมาชิกในแถวและคอลัมน์ให้ถูกต้อง ก็จะสามารถเข้าถึงข้อมูลที่ต้องการได้

การส่งตัวแปรชุด 2 มิติไปเป็นอาร์กิวเมนต์ของฟังก์ชัน จะต้องประกาศขนาดความยาวของคอลัมน์ ดังตัวอย่างฟังก์ชันการกำหนดค่าเริ่มต้นให้กับตัวแปรชุด 2 มิติ แสดงดังตัวอย่างที่ 6.17



**ตัวอย่างที่ 6.17** โปรแกรมเพื่อสร้างข้อมูลเข้าไปเก็บยังตัวแปรชุด 2 มิติที่มีขนาดแถวและคอลัมน์เท่ากัน และให้พิมพ์ค่าดังกล่าว ซึ่งมีลักษณะดังรูป

```

+ + + + +
+ - - - +
+ - - - +
+ - - - +
+ + + + +

```

---

```

#include <stdio.h>
#define MAX 5
void createArray(char [][MAX]);
void printArray(char [][MAX]);
void main() {
 char arr[MAX][MAX];
 createArray(arr);
 printArray(arr);
}
void createArray(char parr[][MAX]) {
 int i, j;
 for (i=0; i<MAX; i++) {
 for (j=0; j<MAX; j++) {
 if (i==0 || j==0 || i==MAX-1 || j==MAX-1)
 parr[i][j] = '+';
 else
 parr[i][j] = '-';
 }
 }
}
void printArray(char parr[][MAX]) {
 int i, j;
 printf("\nShow array : \n");
 for (i=0; i<MAX; i++) {
 for (j=0; j<MAX; j++) {
 printf("%4c", parr[i][j]);
 }
 printf("\n");
 }
}

```

---

จากตัวอย่างจะต้องพิจารณาความสัมพันธ์ที่เกิดขึ้นจากรูปที่โจทย์กำหนด โดยแทนตำแหน่งของสมาชิกลงในรูปจะได้ดังรูปที่ 6.11

|     |     |     |     |     |
|-----|-----|-----|-----|-----|
| 0,0 | 0,1 | 0,2 | 0,3 | 0,4 |
| 1,0 | 1,1 | 1,2 | 1,3 | 1,4 |
| 2,0 | 2,1 | 2,2 | 2,3 | 2,4 |
| 3,0 | 3,1 | 3,2 | 3,3 | 3,4 |
| 4,0 | 4,1 | 4,2 | 4,3 | 4,3 |

รูปที่ 6.11 แสดงการแทนตำแหน่งลงในรูปที่ระบุ

เนื่องจากข้อมูลที่แสดงเป็นเครื่องหมาย + และ - เพราะฉะนั้นจึงประกาศให้ตัวแปรชุดมีเป็นชนิด char ตำแหน่งที่เป็น + จะตรงกับตำแหน่งที่เป็นสีทึบ ให้ i แทนค่าแถวและ j แทนค่าคอลัมน์ จะพบว่าด้านที่เป็นสีทึบนั้นคือ ด้านที่ i มีค่าเป็น 0 (แนวนอนด้านบนสุด) ด้านที่ i มีค่าเท่ากับขนาดตัวแปรชุดลบไป 1 (แนวนอนด้านล่างสุด) ด้านที่ j มีค่าเท่ากับ 0 (แนวตั้งด้านซ้ายสุด) และด้านที่ j มีค่าเท่ากับขนาดตัวแปรชุดลบไป 1 (แนวตั้งด้านขวาสุด) ส่วนตำแหน่งอื่น ๆ จะมีค่าเป็น - โดยอาศัยการสังเกตค่า i และ j ดังกล่าวจะสามารถทำให้เลือกกำหนดค่าสมาชิกของตัวแปรชุดที่ต้องการได้ ถึงแม้ขนาดของรูปจะเปลี่ยนไป ก็จะได้ผลลัพธ์ที่เหมือนเดิม ให้ทดลองโดยการกำหนดค่า MAX ขึ้นใหม่ และทดลองรันโปรแกรมอีกครั้ง

ส่วนการส่งผ่านตัวแปรชุดไปในฟังก์ชันสามารถทำได้ดังตัวอย่าง เช่น หากในฟังก์ชัน main() มีตัวแปรชุดชื่อ arr ดังคำสั่ง

```
char arr[MAX][MAX];
createArray(arr);
```

การส่งตัวแปรชุด arr ไปยังฟังก์ชัน createArray() สามารถอ้างชื่อของตัวแปรชุด arr โดยตรง ซึ่งจะเหมือนกับการอ้างในลักษณะพอยน์เตอร์ และการอ้างถึงตัวแปรชุดมีทีเดียว การประกาศฟังก์ชัน createArray() สามารถทำได้ด้วยคำสั่ง

```
void createArray(char parr[][MAX])
```

เมื่อ parr คือชื่อตัวแปรตัวแปรชุดที่มารับค่าตัวแปรชุด arr โดยที่ไม่ต้องระบุขนาดแถว แต่ให้ระบุขนาดของคอลัมน์เสมอ การอ้างถึงตัวแปร parr เช่น การกำหนดค่า หรือการรับค่า จะเป็นเหมือนกับการอ้างถึงตัวแปรชุด arr ในฟังก์ชัน main() เมื่อค่าใน parr เปลี่ยนแปลง ค่า arr จะเปลี่ยนแปลงด้วยทันที การใช้งานตัวแปรชุด 2 มิติ มักจะใช้คำสั่งวนรอบซ้อนกันในการทำงานเสมอ ดังตัวอย่างเพิ่มเติมในตัวอย่างที่ 6.18 และ 6.19

**ตัวอย่างที่ 6.18** ข้อมูลค่าจ้างของพนักงานในบริษัทแห่งหนึ่ง ซึ่งแบ่งพนักงานออกเป็น 5 ระดับ ในแต่ละระดับจะแบ่งออกเป็น 4 ขั้น ซึ่งแต่ละขั้นจะได้รับเงินเดือนไม่เท่ากัน แสดงดังตารางค่าจ้าง

|         | ขั้น 1 | ขั้น 2 | ขั้น 3 | ขั้น 4 |
|---------|--------|--------|--------|--------|
| ระดับ 1 | 4,200  | 4,400  | 4,650  | 4,950  |
| ระดับ 2 | 5,200  | 5,400  | 5,650  | 5,950  |
| ระดับ 3 | 6,200  | 6,400  | 6,650  | 6,950  |
| ระดับ 4 | 7,200  | 7,400  | 7,650  | 7,950  |
| ระดับ 5 | 8,200  | 8,400  | 8,650  | 8,950  |

ให้เขียนโปรแกรมซึ่งมีการกำหนดค่าเริ่มต้นของค่าจ้างเหล่านี้ในโปรแกรม โดยที่โปรแกรมจะต้องมีความสามารถต่าง ๆ ดังนี้

- คำนวณหาค่าจ้างเฉลี่ยในแต่ละระดับ
- คำนวณหาค่าจ้างเฉลี่ยในแต่ละขั้น
- คำนวณหาค่าจ้างเฉลี่ยของพนักงานทุกระดับและทุกขั้นรวมกัน
- ต้องการเพิ่มค่าจ้างให้พนักงานทุกระดับและขั้นขึ้น 7.5 % ให้แสดงผลลัพธ์ของตารางค่าจ้างที่เพิ่มขึ้น

```
#include <stdio.h>
```

```
#define NO_LEVEL 5
```

```
#define NO_STEP 4
```

```
void calAvgLevel(float [] [NO_STEP]);
```

```
void calAvgStep(float [] [NO_STEP]);
```

```
void calAvgAll(float [] [NO_STEP]);
```

```
void increaseSalary(float [] [NO_STEP], float [] [NO_STEP], float);
```

```
void printSalary(float [] [NO_STEP]);
```

```
void main() {
```

```
 float salary[NO_LEVEL][NO_STEP]={{4200.0, 4400.0, 4650.0, 4950.0}, {5200.0, 5400.0, 5650.0, 5950.0},
 {6200.0, 6400.0, 6650.0, 6950.0}, {7200.0, 7400.0, 7650.0, 7950.0},
 {8200.0, 8400.0, 8650.0, 8950.0}};
```

```
 float newSalary[NO_LEVEL][NO_STEP];
```

```
 calAvgLevel(salary);
```

```
 calAvgStep(salary);
```

```
 calAvgAll(salary);
```

```
 increaseSalary(salary, newSalary, 7.5);
```

```

printf("\n\nNew salary table is\n");
printSalary(newSalary);
}
void calAvgLevel(float sal[][NO_STEP]) {
 float sum, avg;
 int i, j;
 printf("\n\nAverage salary for level : ");
 for (i=0; i<NO_LEVEL; i++) {
 sum = 0.0;
 for (j=0; j<NO_STEP; j++)
 sum = sum + sal[i][j];
 avg = sum / NO_STEP;
 printf("\n\t\t%d is %.2f", i, avg);
 }
}
void calAvgStep(float sal[] [NO_STEP]) {
 float sum, avg;
 int i, j;
 printf("\n\nAverage salary for step : ");
 for (i=0; i<NO_STEP; i++) {
 sum = 0.0;
 for (j=0; j<NO_LEVEL; j++)
 sum = sum + sal[j][i];
 avg = sum / NO_LEVEL;
 printf("\n\t\t%d is %.2f", i, avg);
 }
}
void calAvgAll(float sal[] [NO_STEP]) {
 float sum=0.0, avg;
 int i, j;
 for (i=0; i<NO_LEVEL; i++)
 for (j=0; j<NO_STEP; j++)
 sum = sum + sal[i][j];
 avg = sum / (NO_LEVEL * NO_STEP);
 printf("\n\nAverage salary for all levels and steps is %.2f", avg);
}

```

```

void increaseSalary(float baseSal[][NO_STEP], float newSal[][NO_STEP], float percentInc) {
 int i, j;
 for (i=0; i<NO_LEVEL; i++)
 for (j=0; j<NO_STEP; j++)
 newSal[i][j] = baseSal[i][j] + (baseSal[i][j]*percentInc/100);
}
void printSalary(float sal[][NO_STEP]) {
 int i, j;
 for (i=0; i<NO_LEVEL; i++) {
 for (j=0; j<NO_STEP; j++)
 printf("%12.2f", sal[i][j]);
 printf("\n");
 }
}

```

ปัญหาหลักเมื่อพบเขียนโปรแกรมในลักษณะตัวแปรชุด 2 มิติ คือ ไม่รู้ว่าจะต้องเขียนคำสั่ง for ของแถวหรือคอลัมน์ไว้ที่ตำแหน่งใด สังเกตได้ว่า

- ถ้าเป็นการเข้าถึงสมาชิกทุกตัวจะเขียนคำสั่ง for ของแถวหรือคอลัมน์ก่อนก็ได้ เช่น การคำนวณหาค่าเฉลี่ยของค่าจ้างในทุกระดับและทุกชั้น จะต้องมีการหาผลรวมของค่าจ้างของสมาชิกทุกตัวในตัวแปรชุด ซึ่งจะต้องเข้าถึงสมาชิกทุกตัว เพราะฉะนั้นจะเขียนคำสั่งวนซ้ำของแถวหรือคอลัมน์ก่อนก็ได้ ดังตัวอย่างในฟังก์ชัน calAvgAll( ) จะต้องมีการหาผลรวมทั้งหมดก่อนหาค่าเฉลี่ย และฟังก์ชัน increaseSalary( ) ที่จะต้องมีการเพิ่มค่าจ้างให้ในทุกระดับและทุกชั้น
- หากต้องการหาผลรวมในแนวแถว จะต้องตีความปัญหาให้ได้ก่อน เช่น หาค่าจ้างเฉลี่ยในแต่ละระดับ ในที่นี้ระดับคือแถว เพราะฉะนั้นต้องใช้คำสั่ง for ควบคุมแถวอยู่ดูปนอก ต้องมีการกำหนดให้ค่าตัวแปร sum เป็น 0 ก่อน เพื่อให้ผลรวมในแต่ละชั้นเริ่มต้นที่ 0 เสมอ ในแต่ละแถวจึงจะมีการหาผลรวมของแต่ละชั้น (คอลัมน์) เมื่อได้ผลรวมแต่ละชั้นในระดับนั้นแล้ว จึงจะหาค่าเฉลี่ยในระดับนั้นได้ ดังตัวอย่างในฟังก์ชัน calAvgLevel( ) ในคำสั่ง for ด้านในสังเกตว่าค่า i ที่เป็นค่าแถวจะคงที่เสมอ
- หากต้องการหาผลรวมในแนวคอลัมน์ เช่น หาค่าจ้างเฉลี่ยในแต่ละชั้น ในที่นี้ชั้นเป็นคอลัมน์ ในการคำนวณจะต้องหาว่าผลรวมในแต่ละระดับในชั้นนั้นมีอะไรบ้าง เพราะฉะนั้นคำสั่ง for ควบคุมคอลัมน์จึงอยู่ด้านนอก แล้วจึงหาผลรวมของแต่ละระดับ ก็จะได้ค่าผลรวมเพื่อนำมาหาค่าเฉลี่ยในชั้นนั้นๆ ดังตัวอย่างในฟังก์ชัน calAvgCol( ) ให้สังเกตว่าในคำสั่ง for ด้านในค่าของ i ที่เป็นค่าของคอลัมน์จะคงที่เสมอ

**ตัวอย่างที่ 6.19** โปรแกรมเพื่อรับค่าข้อมูลคะแนนของนักศึกษาในกระบวนวิชาหนึ่งซึ่งมีนักศึกษา 25 คน แบ่งคะแนนของนักศึกษาออกเป็น 5 ส่วน ได้แก่ คะแนนทดสอบย่อย 3 ครั้ง คะแนนสอบกลางภาค และคะแนนสอบปลายภาค ให้คำนวณหาคะแนนรวมที่นักศึกษาแต่ละคนได้รับ และเกรดที่นักศึกษาแต่ละคนจะได้รับเมื่อกำหนดให้

| ช่วงคะแนน   | เกรดที่ได้รับ |
|-------------|---------------|
| น้อยกว่า 50 | F             |
| 50 ถึง 59   | D             |
| 60 ถึง 69   | C             |
| 70 ถึง 79   | B             |
| 80 ขึ้นไป   | A             |

---

```

#include <stdio.h>
#define MAX_STD 3
#define MAX_SCR 2

void readScore(float[][MAX_SCR]);
void findTotalScore(float[][MAX_SCR], float[]);
void calGrade(float[], char[]);
void printResult(float[][MAX_SCR], float[], char[]);

void main() {
 float score[MAX_STD][MAX_SCR], totalScore[MAX_STD];
 char grade[MAX_STD];
 readScore(score);
 findTotalScore(score, totalScore);
 calGrade(totalScore, grade);
 printResult(score, totalScore, grade);
}

void readScore(float pScore[][MAX_SCR]) {
 int i, j;
 for (i=0; i<MAX_STD; i++) {
 printf("\nEnter score for student no.%d : \n", i+1);
 for (j=0; j<MAX_SCR; j++) {
 printf("\tscore no.%d : ", j+1);
 scanf("%f", &pScore[i][j]);
 }
 }
}

```

```

void findTotalScore(float pScore[][MAX_SCR], float pTotal[]) {
 int i, j;
 for (i=0; i<MAX_STD; i++) {
 pTotal[i] = 0.0;
 for (j=0; j<MAX_SCR; j++)
 pTotal[i] = pTotal[i] + pScore[i][j];
 }
}

void calGrade(float pTotal [], char pGrade[]) {
 int i;
 for (i=0; i<MAX_STD; i++) {
 if (pTotal[i] < 50)
 pGrade[i] = 'F';
 else if (pTotal[i] < 60)
 pGrade[i] = 'D';
 else if (pTotal[i] < 70)
 pGrade[i] = 'C';
 else if (pTotal[i] < 80)
 pGrade[i] = 'B';
 else
 pGrade[i] = 'A';
 }
}

void printResult(float pScore[][MAX_SCR], float pTotal[], char pGrade[]) {
 int i, j;
 printf("\nStudent score report\n");
 for (i=0; i<MAX_STD; i++) {
 printf("\t%-2d", i+1);
 for (j=0; j<MAX_SCR; j++)
 printf("%8.2f", pScore [i][j]);
 printf("%8.2f %c\n", pTotal[i], pGrade[i]);
 }
}

```

---

ให้สังเกตการรับข้อมูลเข้ามายังตัวแปรชุดหากเป็นการอ้างชื่อของสมาชิกในตัวแปรชุด จะต้องใช้เครื่องหมาย & เพื่อบอกให้รู้ว่าต้องการอ่านข้อมูลเข้ามาเก็บยังสมาชิกตัวที่ระบุ ทั้งนี้ใช้ได้กับข้อมูลพื้นฐานทุกประเภท ยกเว้นข้อมูลประเภทสตริงที่สามารถอ้างชื่อสมาชิกของตัวแปรชุดโดยไม่จำเป็นต้องใช้เครื่องหมาย &

## 9. คอมมมานไลน์อาร์กิวเมนต์ (Command-line Arguments)

การใช้งานโปรแกรมที่เขียนด้วยภาษาที่สามารถส่งอาร์กิวเมนต์ให้กับโปรแกรมผ่านทางคำสั่งที่เรียกใช้งานโปรแกรมนั้น อาร์กิวเมนต์ที่ส่งผ่านจากคำสั่งที่เรียกใช้งานโปรแกรมจะถูกส่งเข้ามาเป็นอาร์กิวเมนต์ของฟังก์ชันแรกๆ ที่ทำงานของโปรแกรม ซึ่งทุกโปรแกรมจะเริ่มทำงานที่ฟังก์ชัน `main ( )` เสมอ ตัวอย่างเช่น มีโปรแกรมชื่อ `echo` ต้องการให้โปรแกรมนี้ทำหน้าที่รับข้อความความยาวไม่จำกัด แล้วแสดงข้อความนั้นออกทางจอภาพ การเรียกใช้งานโปรแกรมจะใช้คำสั่งในลักษณะคอมมมานไลน์ดังนี้

```
echo hello, world
```

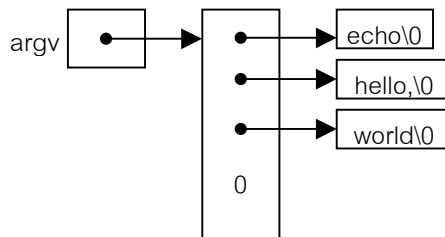
จากตัวอย่างจะเห็นว่ามีการรับข้อความที่ต้องการแสดงผล 2 ข้อความ คือ `hello` และ `world` ผลลัพธ์การทำงานของโปรแกรมนี้คือ

```
hello, world
```

การประกาศฟังก์ชัน `main ( )` จะต้องการมีประกาศอาร์กิวเมนต์เพื่อรับข้อความที่จะส่งเข้ามา โดยปกติอาร์กิวเมนต์ที่ส่งเข้ามาจะเป็นในลักษณะของข้อความเท่านั้น หากผู้ใช้ต้องการนำค่าไปคำนวณเช่น รับค่าตัวเลขเข้ามาทางคอมมมานไลน์อาร์กิวเมนต์ จะต้องเรียกใช้ฟังก์ชันในการแปลงข้อความเป็นตัวเลขด้วยตนเอง การประกาศฟังก์ชันที่รองรับคอมมมานไลน์อาร์กิวเมนต์ที่มีรูปแบบมาตรฐานดังนี้

```
main (int argc, char *argv[])
```

`argc` จะบอกถึงจำนวนอาร์กิวเมนต์ที่ส่งเข้ามารวมทั้งชื่อโปรแกรมด้วย ในที่นี้จะได้ `argc` คือ 3 ส่วน `argv` เป็นข้อมูลพอยน์เตอร์ชี้ไปยังตัวแปรชุดตัวอักษร ซึ่งความยาวไม่จำกัดและไม่จำกัดจำนวนของอาร์กิวเมนต์ที่จะส่งเข้ามา จะได้ข้อมูลใน `argv` ดังนี้ `argv[0]` คือ `echo` `argv[1]` คือ `hello,` และ `argv[2]` คือ `world` สามารถจำลองรูปของการเก็บข้อมูลดังรูปที่ 6.12 แสดงค่าที่ได้รับจากคอมมมานไลน์อาร์กิวเมนต์ดังตัวอย่างที่ 6.20



รูปที่ 6.12 แสดงแบบจำลองของ `argv` ในการเรียกใช้โปรแกรม `echo`



### ตัวอย่างที่ 6.20 โปรแกรมแสดงผลข้อความที่รับจากการเรียกใช้งานโปรแกรม

---

```
#include <stdio.h>

int main (int argc, char *argv[]){
 int i;
 for (int i = 1; i < argc; i++)
 printf("%s ", argv[i]);
 return 0;
}
```

---

เนื่องจาก argv เป็นตัวแปรพอยน์เตอร์ชี้ไปยังตัวแปรชุด การอ้างถึง argv สามารถทำได้ในลักษณะของพอยน์เตอร์เช่นเดียวกัน ดังตัวอย่างที่ 6.21

### ตัวอย่างที่ 6.21 โปรแกรมแสดงผลข้อความที่รับจากการเรียกใช้โปรแกรมในลักษณะพอยน์เตอร์

---

```
#include <stdio.h>

int main (int argc, char *argv[]){
 while (--argc > 0)
 printf ("%s%s", *++argv, (argc > 1) ? " : ");
 return 0;
}
```

---

การทำงานคำสั่ง \*++argv ในรอบแรกจะมีการเลื่อนพอยน์เตอร์ชี้จากอาร์กิวเมนต์ตัวที่ 0 ไปยังอาร์กิวเมนต์ตัวที่ 1 แล้วอ่านข้อมูลขึ้นมาแสดงจะได้ hello, ส่วนในรอบที่ 2 จะเลื่อนพอยน์เตอร์ชี้ไปยังสมาชิกตัวที่ 2 คือ world แล้วแสดงผล การใช้งานคอมมานไลน์อาร์กิวเมนต์นั้นค่าที่รับเข้าจะเป็นข้อมูลสตริงเสมอ หากป้อนข้อมูลเข้าเป็นตัวเลขจะต้องมีการแปลงค่าข้อมูลนั้นก่อนนำไปใช้ทุกครั้ง แสดงดังตัวอย่างที่ 6.22

**ตัวอย่างที่ 6.22** โปรแกรมแสดงการสลับค่าตัวเลขจำนวนเต็ม 2 จำนวน โดยใช้คอมมานไลน์อาร์กิวเมนต์

---

```
#include <stdio.h>
void swap (int *, int *);
int main (int argc, char *argv[]) {
 int a, b;
 if (argc < 3) {
 printf("\nIncorrect input format: <program> <int> <int>");
 exit(-1);
 }
 a = atoi(argv[1]);
 b = atoi(argv[2]);
 printf("\nBefore swap A = %d, B = %d", a, b);
 swap(&a, &b);
 printf("\nAfter swap A = %d, B = %d", a, b);
 return 0;
}
void swap (int *x, int *y) {
 int temp;
 temp = *x;
 *x = *y;
 *y = temp;
}
```

---

**แบบฝึกหัดบทที่ 6**

- เขียนโปรแกรมเพื่อรับข้อมูล N จำนวน โดยเก็บข้อมูลในตัวแปรชุด และให้พิมพ์ข้อมูลในลำดับที่กลับกับการป้อน ข้อมูล เช่น ป้อนข้อมูล 1 2 3 4 5 6 จะพิมพ์คำตอบในลักษณะ 6 5 4 3 2 1
- เขียนโปรแกรมเพื่อรับข้อมูลเข้ามาเก็บในตัวแปรชุด 2 ชุด คือ  $x_i$  และ  $y_i$  แต่ละชุดมีข้อมูลเท่ากันคือ N ตัว ให้หาค่า

$$\sum_{i=1}^N [(x_i - y_i)^2 / y_i]$$

- เขียนโปรแกรมเพื่อรับข้อมูลจำนวนเต็มเข้ามาเก็บยังตัวแปรชุด 2 ตัวแปรชุด ซึ่งมีขนาดเท่ากัน คือ N โดยมีข้อกำหนดในการรับ ข้อมูลของตัวแปรชุดทั้งคู่ คือ ข้อมูลตัวแรกจะเป็นข้อมูลอะไรก็ได้แต่จะต้องมีค่าอยู่ในช่วง 1 ถึง 20 ส่วนข้อมูลตัวถัดไปต้องมีค่ามากกว่าสามารถตัวก่อนหน้าตัวมันเอง โดยมีความมากกว่าไม่เกิน 5 เมื่อรับข้อมูลเสร็จแล้ว ให้ทำการรวม (Merge) ตัวแปรชุดทั้งคู่เข้าด้วยกัน โดยเรียงตามลำดับข้อมูลจากน้อยไปมาก หากมีค่าเท่ากันจะนำค่าใดไปไว้ในตัวแปรชุดก่อนก็ได้ เช่น

|             |   |   |   |    |    |             |   |   |   |    |    |
|-------------|---|---|---|----|----|-------------|---|---|---|----|----|
| ตัวแปรชุด 1 | 1 | 3 | 7 | 12 | 14 | ตัวแปรชุด 2 | 2 | 3 | 6 | 11 | 14 |
|-------------|---|---|---|----|----|-------------|---|---|---|----|----|

|                |   |   |   |   |   |   |    |    |    |    |
|----------------|---|---|---|---|---|---|----|----|----|----|
| ตัวแปรชุดคำตอบ | 1 | 2 | 3 | 3 | 6 | 7 | 11 | 12 | 14 | 14 |
|----------------|---|---|---|---|---|---|----|----|----|----|

- เขียนโปรแกรมโดยใช้ตัวแปรชุด เพื่อรับข้อมูลชื่อ และคะแนนสอบของนักเรียนห้องหนึ่งซึ่งมี 50 คน โดยรับข้อมูลเป็นจำนวนเต็ม โปรแกรมจะต้องมีความสามารถต่าง ๆ ดังนี้
  - หาคะแนนเฉลี่ยของนักเรียนทั้งห้อง
  - พิมพ์ชื่อและคะแนนของนักเรียนที่ได้คะแนนต่ำกว่า 50 คะแนน
  - คำนวณค่าความแปรปรวนจากสูตร

$$\text{Variant} = \sum_{i=1}^N \frac{(x_i - \bar{x})^2}{N - 1}$$

โดยที่  $x_i$  แทน คะแนนของนักเรียนแต่ละคน

$\bar{x}$  แทน คะแนนเฉลี่ย

N แทน จำนวนนักเรียนทั้งหมด

- พิมพ์กราฟความถี่ของคะแนนซึ่งมีค่าตั้งแต่ 0 ถึง 100 โดยใช้เครื่องหมาย \* แทนจำนวนความถี่ของคะแนน โดยแสดงผลทีละ 20 คะแนน เช่น

0 - \*\*\* 3

-> แสดงว่ามีคนได้ 0 คะแนนจำนวน 3 คน

- เขียนโปรแกรมเพื่อรับข้อความจากผู้ใช้ ให้แปลงข้อความดังกล่าวโดย ตัวอักษรใดเป็นตัว a หรือ A ให้แทรกข้อความแทนด้วย 555 ตัวอักษรใดเป็นตัว u หรือ U ให้แทนที่ด้วย Hanaga และแสดงผลที่ได้
- เขียนโปรแกรมเพื่อรับอักขระทางแป้นพิมพ์ทีละ 1 ตัวอักษร จากนั้นทำการนับจำนวนตัวอักษร A-Z และ a-z ที่รับเข้าไป ข้อมูลตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็กจะนับเป็นอักษรตัวเดียวกัน โดยจะสิ้นสุดการรับข้อมูลเมื่อผู้ใช้กด "%" จากนั้นทำการพิมพ์ความถี่ของจำนวนแต่ละตัวอักษร A-Z หรือ a-z ที่ป้อนเข้าไป กำหนดให้ทำการพิมพ์ "\*" เท่ากับจำนวนค่าความถี่ของการรับแต่ละตัวอักษรที่นับได้
- เขียนโปรแกรมเพื่อเก็บข้อมูลต่อไปนี้ลงในตัวแปรชุด และแสดงผลข้อมูลที่เก็บภายในตัวแปรชุดนั้น ข้อมูลมีดังนี้
 

|   |   |   |   |   |
|---|---|---|---|---|
| ( | O | O | O | ) |
| ( | + | Y | Y | ) |
| ( | X | + | Y | ) |
| ( | X | X | + | ) |
| ( | O | O | O | ) |
- เขียนโปรแกรมเพื่อรับค่าข้อมูลการวัดอุณหภูมิร่างกายของคนใช้รายหนึ่งเป็นเวลา 7 วัน โดยแต่ละวันจะทำการวัดอุณหภูมิ 4 ครั้งในเวลาที่ตั้งกันคือ 6.00 น. 12.00 น. 18.00 น. และ 24.00 น. เขียนโปรแกรมเพื่อทำการเก็บข้อมูล โปรแกรมจะต้องมีความสามารถ
  - คำนวณอุณหภูมิสูงสุดและต่ำที่สุดที่วัดได้ในแต่ละวัน
  - คำนวณอุณหภูมิเฉลี่ยที่วัดได้ในแต่ละวัน ทั้ง 7 วัน
  - คำนวณอุณหภูมิเฉลี่ยที่ได้ในแต่ละช่วงเวลา ทั้ง 4 ช่วงเวลา
  - คำนวณอุณหภูมิเฉลี่ยรวมทั้งหมด
- เขียนโปรแกรมเพื่อเก็บข้อมูลประตูได้และเสียในการแข่งขันฟุตบอลรายการหนึ่ง ซึ่งมีทีมฟุตบอลเข้าร่วมแข่งขัน 4 ทีม โดยแต่ละทีมจะทำการแข่งขันแบบพบกันหมด (เล่นทีมละ 3 ครั้ง) ให้เขียนโปรแกรมโดยแยกประตูได้และเสียออกเป็นตัวแปรชุด 2 มิติจำนวน 2 ตัวแปรชุด โปรแกรมจะทำหน้าที่
  - รับข้อมูลประตูได้และเสียของแต่ละทีม
  - หาผลต่างประตูได้เสียในการแข่งขันแต่ละครั้งของแต่ละทีม และให้แสดงทีมที่มีผลต่างดีที่สุด
  - หาผลรวมประตูที่ได้ของแต่ละทีม และให้แสดงทีมที่ทำประตูได้มากที่สุด
  - หาประตูได้และเสียเฉลี่ยของแต่ละทีม

10. เขียนโปรแกรมเพื่อสร้างตารางสรุปจำนวนนักศึกษาโดยแยกตามคณะและชั้นปีที่นักศึกษาสังกัด ดังตาราง

|           | First Year | Second Year | Third Year | Fourth Year | >4  | Total |
|-----------|------------|-------------|------------|-------------|-----|-------|
| Faculty 1 | 1500       | 1200        | 1150       | 1100        | 20  | 4970  |
| Faculty 2 | 3500       | 3300        | 3200       | 3100        | 100 | 12200 |
| ...       |            |             |            |             |     |       |
| Total     | ...        | ...         | ...        | ...         | ... | ...   |

โปรแกรมจะรับข้อมูลจำนวนนักศึกษาซึ่งแยกตามคณะและชั้นปี ให้เก็บข้อมูลโดยใช้ตัวแปรชุด 2 มิติ หลังจากนั้นให้สร้างตารางสรุปจำนวนนักศึกษา โดยที่มีการสรุปยอดรวมของนักศึกษาแยกตามคณะ และชั้นปี รวมทั้งสรุปยอดนักศึกษารวมด้วย

- เขียนโปรแกรมเพื่อหาค่าผลบวกของเมตริกซ์ 2 เมตริกซ์ที่มีขนาดมิติเท่ากัน คือ  $3 \times 2$  ให้รับข้อมูลของเมตริกซ์ทั้ง 2 และคำนวณค่าผลบวกของเมตริกซ์ที่ได้
- เขียนโปรแกรมเพื่อหาผลคูณของเมตริกซ์ 2 เมตริกซ์ เมตริกซ์แรกมีขนาด  $N \times M$  เมตริกซ์ที่ 2 มีขนาด  $M \times N$  ให้หาผลคูณของเมตริกซ์ทั้ง 2 ซึ่งมีขนาดของเมตริกซ์คำตอบเท่ากับ  $N \times N$
- เขียนโปรแกรมเพื่อรับข้อมูลเมตริกซ์ขนาด  $N \times N$  แล้วให้ตรวจสอบว่าเมตริกซ์นั้นเป็น Upper-triangular Matrix หรือ Lower-triangular Matrix หรือไม่ โดยที่ Triangular Matrix คือ เมตริกซ์ขนาด  $N \times N$  ที่มีการแบ่งข้อมูลออกเป็น 2 ส่วนตามเส้นแนวทแยง คือ เป็นด้านบนและด้านล่าง หากด้านบนมีค่าเป็น 0 ทั้งหมด เรียกว่า Upper-triangular Matrix หากด้านล่างมีค่าเป็น 0 ทั้งหมด เรียกว่า Lower-triangular Matrix ตัวอย่างเช่น

Upper-triangular Matrix

|   |   |   |
|---|---|---|
| a | 0 |   |
| b | c |   |
| d | e | f |

Lower-triangular Matrix

|   |   |   |
|---|---|---|
| a | b |   |
| 0 | c |   |
| 0 | d | e |
| 0 | 0 | f |

# โครงสร้างและยูเนียน

## ( Structures and Unions )

7

นอกจากตัวแปรชุดซึ่งเป็นข้อมูลที่ประกอบด้วยชุดของข้อมูลชนิดเดียวกันหลาย ๆ ตัว ภาษาซียังมีชนิดข้อมูลที่ประกอบขึ้นจากข้อมูลประเภทพื้นฐานอีกประเภทหนึ่ง ข้อมูลแบบโครงสร้าง (Structure) และข้อมูลแบบยูเนียน เป็นชนิดข้อมูลที่ประกอบด้วยตัวแปรตั้งแต่ 1 ตัวขึ้นไป ซึ่งไม่จำเป็นต้องเป็นข้อมูลประเภทเดียวกัน มักจะใช้เมื่อต้องการเก็บข้อมูลที่เป็นกลุ่มเดียวกัน เช่น ข้อมูลของพนักงาน จะประกอบด้วยข้อมูลรหัสพนักงาน ชื่อ ที่อยู่ เป็นต้น

### 1. ความรู้ทั่วไปเกี่ยวกับโครงสร้าง

ในชีวิตประจำวันของคนทั่วไปมักจะเกี่ยวข้องกับข้อมูลเสมอ ๆ ลักษณะของข้อมูลที่พบเห็นในชีวิตประจำวันมักจะเป็นข้อมูลที่พิจารณาเป็นกลุ่มของข้อมูล เช่น เมื่อพิจารณาข้อมูลของคนหนึ่งคน อาจพิจารณาคุณสมบัติในเรื่องของชื่อ นามสกุล อายุ ความสูง น้ำหนัก พิจารณารถยนต์หนึ่งคันอาจจะพิจารณาถึงยี่ห้อ รุ่น หมายเลขทะเบียนรถ สี เป็นต้น จะเห็นว่าเวลาที่พบเห็นข้อมูลหนึ่ง ๆ ข้อมูลเหล่านั้นมักจะประกอบด้วยคุณสมบัติหรือข้อมูลย่อยเสมอ กรณีของข้อมูลโครงสร้างก็เป็นการพิจารณาข้อมูลในลักษณะเดียวกัน เป็นการประกาศชนิดข้อมูลชนิดใหม่ การประกาศประเภทข้อมูลโครงสร้างขึ้นมาชนิดหนึ่ง จะต้องพิจารณาว่าข้อมูลโครงสร้างนั้นประกอบไปด้วยคุณสมบัติอะไรบ้าง และจะต้องมีการประกาศคุณสมบัติของข้อมูลโครงสร้างคู่กันเสมอ

รูปแบบของการประกาศข้อมูลโครงสร้าง

```
struct <ชื่อโครงสร้าง> {
 <คุณสมบัติ 1> ;
 <คุณสมบัติ 2> ;
};
```

สมาชิก

struct เป็นคำสั่งที่ใช้ในการประกาศโครงสร้าง และผู้ใช้จะต้องประกาศชื่อของโครงสร้างนั้น ๆ ภายในโครงสร้างจะประกอบด้วยคุณสมบัติหรือสมาชิกที่ตัวก็ได้ พิจารณาจากตัวอย่างเรื่องของคนหนึ่งคน จะพบว่าคุณสมบัติของคนที่เราสนใจ คือ ชื่อ นามสกุล อายุ ความสูง น้ำหนัก จะประกาศโครงสร้างของคนได้ดังนี้

```

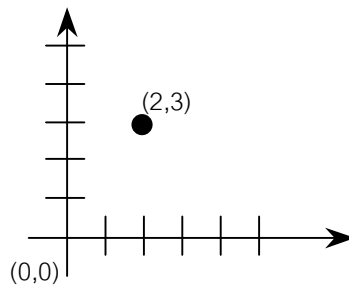
struct man {
 char name[16];
 char surname[21];
 int age;
 float height;
 float weight;
};

```

จะเห็นว่าข้อมูลโครงสร้างเป็นการวางกรอบคุณสมบัติของข้อมูลที่เราพบเห็นให้เป็นระเบียบ เป็นกลุ่มข้อมูลเดียวกัน เพื่อให้สามารถอ้างอิงได้ง่ายขึ้น โดยที่คุณสมบัติข้อมูลภายในโครงสร้างอาจจะประกอบขึ้นจากคุณสมบัติที่มีชนิดข้อมูลที่แตกต่างกัน และจะไม่มีการจำกัดจำนวนคุณสมบัติภายในโครงสร้าง แต่ควรจะจัดเฉพาะคุณสมบัติของโครงสร้างให้เหมาะสมกับชื่อโครงสร้างนั้น ๆ

## 2. การใช้งานตัวแปรโครงสร้าง

พิจารณาจากการเก็บข้อมูลของจุดบนแกนโคออดิเนต ดังรูปที่ 7.1



รูปที่ 7.1 แสดงจุดบนแกนโคออดิเนต

หากต้องการเก็บข้อมูลจุดบนแกนโคออดิเนต โครงสร้างที่จะต้องสร้างคือ จุด ภายในโครงสร้างของจุดจะประกอบด้วยคุณสมบัติของข้อมูลแกน x และ y เป็นข้อมูลจำนวนเต็มชนิด int สามารถประกาศโครงสร้างที่ใช้ดังนี้

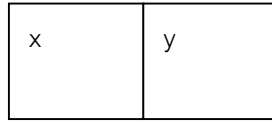
```

struct point {
 int x;
 int y;
};

```

จากตัวอย่างเป็นการประกาศประเภทข้อมูลโครงสร้างที่มีชื่อว่า point ( เรียก point ว่า Structure Tag) ประกอบด้วยคุณสมบัติคือ x และ y เก็บข้อมูลแบบ int ( เรียกว่า Member) การประกาศชื่อสมาชิกภายใน struct จะใช้ชื่อใดก็ได้ว่าจะซ้ำกับชื่อตัวแปรที่อยู่ภายนอก struct แต่ชื่อที่อยู่ภายใน struct เดียวกันห้ามประกาศชื่อซ้ำกัน สามารถจำลองการเก็บข้อมูลของโครงสร้างได้ดังรูปที่ 7.2

struct point



รูปที่ 7.2 แสดงแบบจำลองของโครงสร้าง

## 2.1 รูปแบบการประกาศตัวแปรโครงสร้าง

การประกาศตัวแปรโครงสร้างสามารถทำได้หลายรูปแบบ ซึ่งในที่นี้จะสรุปตัวอย่างให้เห็น 4 รูปแบบ ดังนี้

|                                                                                                                      |                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| <p><u>รูปแบบที่ 1</u></p> <pre> struct {     int x;     int y; } p1,p2;           </pre>                             | <p><u>รูปแบบที่ 2</u></p> <pre> struct point {     int x;     int y; } p1,p2;           </pre>   |
| <p><u>รูปแบบที่ 3</u></p> <pre> struct point {     int x;     int y; }; typedef struct point Point;           </pre> | <p><u>รูปแบบที่ 4</u></p> <pre> typedef struct {     int x;     int y; } point;           </pre> |

- รูปแบบที่ 1 เป็นการประกาศรูปแบบโครงสร้างหนึ่งขึ้นมา พร้อมกับการประกาศตัวแปรสำหรับโครงสร้างนี้ 2 ตัว คือ p1 และ p2
- รูปแบบที่ 2 เป็นการประกาศรูปแบบโครงสร้างและตั้งชื่อรูปแบบนี้ว่า point พร้อมกับการทำการประกาศตัวแปรสำหรับโครงสร้างนี้ 2 ตัว คือ p1 และ p2 การประกาศโครงสร้างในรูปแบบนี้สามารถนำโครงสร้างนี้ไปใช้สำหรับการประกาศตัวแปรใหม่ที่มีโครงสร้างเหมือนกับ p1 และ p2 ได้โดยไม่ต้องประกาศโครงสร้างนี้ใหม่ โดยหากต้องการประกาศตัวแปร p3 ซึ่งมีโครงสร้างรูปแบบเดียวกับ p1 และ p2 สามารถทำได้โดยเพิ่มคำสั่งต่อไปนี้



```
struct point p3;
```

โดยวางคำสั่งนี้ไว้ที่ตำแหน่งหลังคำสั่งประกาศโครงสร้าง point

- รูปแบบที่ 3 และ 4 เป็นการกำหนดชนิดข้อมูลใหม่โดยใช้คำสั่ง typedef เพื่อให้สามารถอ้างถึงรูปแบบโครงสร้างได้สั้นลง จากรูปแบบที่ 2 เมื่อต้องการประกาศตัวแปรใหม่ต้องอ้างถึงโครงสร้างด้วยคำสั่ง

```
struct point point1;
```

หากใช้การประกาศโครงสร้างในรูปแบบที่ 3 หรือ 4 สามารถประกาศตัวแปรใหม่ด้วยคำสั่ง

```
Point point1;
```

สำหรับตัวอย่างโปรแกรมที่ใช้ในหนังสือเล่มนี้ได้เลือกใช้วิธีการประกาศตัวแปรโครงสร้างโดยใช้รูปแบบที่ 4

## 2.2 การประกาศค่าเริ่มต้นให้กับสมาชิกของตัวแปรโครงสร้าง

ในการประกาศตัวแปรพื้นฐานเราสามารถกำหนดค่าเริ่มต้นให้กับตัวแปรนั้นได้ ตัวอย่างเช่น หากต้องการประกาศตัวแปร x มีชนิดเป็น int โดยกำหนดให้มีค่าเริ่มต้นเท่ากับ 3 สามารถทำได้โดยคำสั่ง

```
int x = 3;
```

สำหรับตัวแปรโครงสร้างเราสามารถกำหนดค่าเริ่มต้นให้กับสมาชิกของตัวแปรโครงสร้างนั้นได้เช่นกัน ตัวอย่างเช่น ต้องการสร้างตัวแปรโครงสร้างชื่อ pt ซึ่งเป็นตัวแปรที่มีรูปแบบโครงสร้าง Point สามารถทำได้ดังนี้

```
Point pt = { 320, 200 };
```

ค่า x ของตัวแปรโครงสร้าง pt จะมีค่าเท่ากับ 320 ค่า y จะมีค่าเท่ากับ 200 เราสามารถจำลองการเก็บข้อมูลตัวแปรโครงสร้าง pt ได้ดังรูปที่ 7.3

pt

|     |     |
|-----|-----|
| x   | y   |
| 320 | 200 |

รูปที่ 7.3 แสดงแบบจำลองของตัวแปรโครงสร้าง

## 2.3 การอ้างถึงสมาชิกของตัวแปรโครงสร้าง

การอ้างถึงสมาชิกภายใน struct สามารถทำได้ในรูปแบบต่อไปนี้

```
structure-name . member
```

ตัวอย่างเช่น เมื่อต้องการอ้างถึงสมาชิกภายใน struct ว่าอยู่ตรงกับจุดใดบนแกนโคออดิเนตจะใช้

```
printf ("%d, %d", maxpt.x, maxpt.y);
```

**ตัวอย่างที่ 7.1** โปรแกรมเพื่อรับค่าจุดบนแกนโคออดิเนตของรูปสี่เหลี่ยม และทำการคำนวณหาพื้นที่ของรูปสี่เหลี่ยมนั้น

---

```
#include <stdio.h>
typedef struct {
 int x;
 int y;
} point;
void main() {
 point pt1,pt2 ;
 int area;

 /* Input Data */
 printf("\nEnter rectangle data\n");
 printf("\tStart point x : ");
 scanf("%d", &pt1.x);
 printf("\tStart point y : ");
 scanf("%d", &pt1.y);
 printf("\tEnd point x : ");
 scanf("%d", &pt2.x);
 printf("\tEnd point y : ");
 scanf("%d", &pt2.y);

 area = (pt2.x-pt1.x) * (pt2.y-pt1.y);
 printf("\nArea is%d", area);
}
```

---

**ตัวอย่างที่ 7.2** เขียนโปรแกรมเพื่อรับข้อมูลของผู้ใช้ซึ่งประกอบด้วย ชื่อ นามสกุล ความสูง และน้ำหนัก แล้วให้คำนวณค่าดัชนีน้หนัก (Body Mass Index : BMI) ซึ่งสามารถคิดได้จากสูตร  $BMI = w / h^2$  โดยที่ w แทนน้ำหนักตัวมีหน่วยเป็นกิโลกรัม และ h แทนความสูงมีหน่วยเป็นเมตร หากค่า BMI อยู่ในช่วง 20-25 ให้ขึ้นข้อความว่า "Normal BMI." แต่หากอยู่นอกช่วงดังกล่าวให้ขึ้นข้อความว่า "Dangerous BMI." ให้ใช้โครงสร้างทำหน้าที่เก็บข้อมูลของผู้ใช้ ค่า BMI และผลลัพธ์ที่ได้ และให้แสดงข้อมูลทั้งหมด

---

```

#include <stdio.h>
#include <string.h>
typedef struct {
 char name[16];
 char surname[20];
 float height;
 float weight;
 float bmi;
 char answer[14];
} student;
void main() {
 student std;
 printf("Enter student data\n");
 printf("\tName : ");
 scanf("%s", std.name);
 printf("\tSurname : ");
 scanf("%s", std.surname);
 printf("\tHeight (m) : ");
 scanf("%f", &std.height);
 printf("\tWeight (Kg) : ");
 scanf("%f", &std.weight);

 std.bmi = std.weight / (std.height * std.height);
 if (std.bmi >= 20 && std.bmi <= 25)
 strcpy(std.answer, "Normal BMI");
 else
 strcpy(std.answer, "Dangerous BMI");
 printf("\n\nBMI result");
 printf("\n%s %s weight %.2f kg. height %.2f", std.name, std.surname, std.weight, std.height);
 printf("\n\tBody mass index %.2f is %s", std.bmi, std.answer);
}

```

---

สังเกตที่การรับข้อมูลของข้อมูลสตริง เช่น ชื่อ (std.name) และนามสกุล (std.surname) ไม่ต้องใช้เครื่องหมาย & หน้าตัวแปรนั้น เนื่องจาก name และ surname เป็นตัวแปรชุดตัวอักษร การอ้างถึงชื่อจะเป็นการอ้างถึงแอดเดรส นั้นอยู่แล้ว แต่ข้อมูลอื่นคือ น้ำหนัก (std.weight) และความสูง (std.height) เป็นข้อมูลพื้นฐานจึงต้องมีเครื่องหมาย & นำหน้า

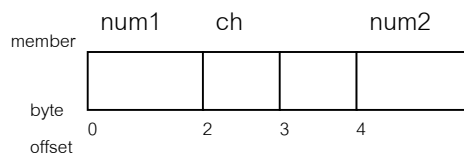
### 3. การเก็บข้อมูลแบบโครงสร้าง

การเก็บข้อมูลแบบโครงสร้างภายในหน่วยความจำจะเก็บตามลำดับที่มีการประกาศสมาชิกของข้อมูลนั้น โดยทั่วไปข้อมูลแบบโครงสร้างจะประกอบขึ้นจากข้อมูลหลาย ๆ ชนิด และข้อมูลแต่ละชนิดมักจะมีการจองพื้นที่ใช้งานแตกต่างกัน เนื่องจากการจองพื้นที่หน่วยความจำในระบบส่วนใหญ่จะจองแอดเดรส (address) ที่หารด้วย 2 หรือ 4 ลงตัว จากตัวอย่าง

```
typedef struct {
 int num1;
 char ch;
 int num2;
} alignment;

alignment example; /* ประกาศตัวแปร */
```

สมมติให้ข้อมูลประเภท int ซึ่งจะต้องใช้แอดเดรสที่หารด้วย 2 หรือ 4 ลงตัว ใช้พื้นที่ 2 ไบต์ในการเก็บข้อมูล ข้อมูลประเภท char จะใช้พื้นที่ที่เลขที่อยู่ใดก็ได้ การจองพื้นที่ในหน่วยความจำของตัวแปร example จะได้ดังรูปที่ 7.5



รูปที่ 7.5 แสดงการจองพื้นที่หน่วยความจำของตัวแปร example

จะเห็นว่า num2 จะไม่สามารถใช้พื้นที่ที่ติดกับ ch ได้ เนื่องจาก num2 เป็นข้อมูลประเภทตัวเลขที่จะต้องใช้แอดเดรสในหน่วยความจำที่หารด้วย 2 หรือ 4 ลงตัว การจองพื้นที่ของตัวแปร example จึงทำให้เกิดที่ว่างที่ไม่สามารถนำมาใช้ประโยชน์ได้ เพราะฉะนั้นการประกาศสมาชิกของโครงสร้างจะมีผลต่อการจองพื้นที่ในหน่วยความจำด้วย

หมายเหตุ

การทำงานของตัวแปรโครงสร้างสามารถทำงานได้เช่นเดียวกับตัวแปรประเภทอื่น ๆ ยกเว้นการเปรียบเทียบตัวแปรโครงสร้างกับตัวแปรโครงสร้างโดยตรง เนื่องจากข้อมูลของตัวแปรโครงสร้างจะเก็บอยู่ในตัวแปรที่เป็นสมาชิกของ struct การเปรียบเทียบจึงต้องทำผ่านตัวแปรที่เป็นสมาชิกของโครงสร้างเท่านั้น

ตัวอย่าง

```
typedef struct {
 float real, img;
} complex;

complex c1 = { 3.01, 4.5 };
complex c2 = { 3.015, 4.45 };
```

หากต้องการตรวจสอบว่าจำนวนเชิงซ้อน c1 และ c2 มีค่าเท่ากันหรือไม่ต้องทำการตรวจสอบโดยใช้คำสั่งดังนี้

```
if ((c1.real == c2.real) && (c1.img == c2.img))
```

...

เราไม่สามารถเปรียบเทียบโดยใช้คำสั่งดังนี้

```
if (c1 == c2)
```

...

#### 4. การใช้ข้อมูลแบบโครงสร้างกับฟังก์ชัน

การใช้งานตัวแปรโครงสร้างกับฟังก์ชันสามารถทำได้หลายลักษณะเช่นเดียวกับการใช้งานตัวแปรพื้นฐาน ทั้งการใช้ฟังก์ชันคืนค่าเป็น struct และการส่งอาร์กิวเมนต์ให้ฟังก์ชันเป็นตัวแปร โครงสร้าง แสดงดังตัวอย่างที่ 7.3

##### ตัวอย่างที่ 7.3 การรับข้อมูลและแสดงผลข้อมูลตัวแปรโครงสร้างโดยใช้ฟังก์ชัน

---

```
#include <stdio.h>

struct point {
 int x;
 int y;
};

struct point readPoint();
void printPoint(struct point);
void main() {
 struct point pt;
 pt = readPoint();
 printPoint(pt);
}

struct point readPoint() {
 struct point p1;
 printf("Enter point x : ");
 scanf("%d", &p1.x);
 printf("Enter point y : ");
 scanf("%d", &p1.y);
 return (p1);
}

void printPoint(struct point p) {
 printf("\n\nPoint x : %d, y : %d", p.x, p.y);
}
```

---

ฟังก์ชันที่มีการคืนค่าเป็นโครงสร้างมีรูปแบบคือ

```
struct <ชื่อโครงสร้าง> <ชื่อฟังก์ชัน> (<ค่าที่ส่งมายังฟังก์ชัน>) {
 struct <ชื่อโครงสร้าง> <ตัวแปรโครงสร้างชั่วคราว>

 return (<ตัวแปรโครงสร้างชั่วคราว>);
}
```

ดังตัวอย่างของฟังก์ชัน readPoint() ที่มีการคืนค่าเป็นโครงสร้าง point

```
struct point readPoint() {
 struct point p1;
 ...
 return (p1);
}
```

ฟังก์ชันที่มีการคืนค่าโครงสร้างมักจะมีตัวแปรโครงสร้างชั่วคราวที่ใช้ทำงานในฟังก์ชันนั้นเสมอ เมื่อฟังก์ชันทำงานเสร็จก็จะทำงานคืนค่าด้วยคำสั่ง return ก็จะทำให้การสำเนาจากตัวแปรโครงสร้างชั่วคราวกลับไปยังตัวแปรโครงสร้างที่เรียกใช้ฟังก์ชันนั้น จากตัวอย่างจะมีการสำเนาจากตัวแปร p1 ในฟังก์ชัน readPoint() กลับไปยังตัวแปร pt ในฟังก์ชัน main() เป็นผลมาจากการเรียกใช้ในฟังก์ชัน main() ด้วยคำสั่ง

```
pt = readPoint();
```

ส่วนฟังก์ชันที่มีการรับอาร์กิวเมนต์เป็นตัวแปรโครงสร้างจากในตัวอย่าง คือ ฟังก์ชัน printPoint() ซึ่งมีการรับค่าตัวแปรโครงสร้าง pt จากฟังก์ชัน main() ซึ่งเรียกใช้ด้วยคำสั่ง

```
printPoint (pt);
```

ในการประกาศฟังก์ชัน printPoint จะต้องประกาศพารามิเตอร์เป็นตัวแปรโครงสร้างเพื่อรับค่าของข้อมูลดังกล่าว ด้วยคำสั่ง

```
void printPoint (struct point p1)
```

เมื่อมีการเรียกใช้ฟังก์ชัน printPoint() จะมีการสำเนาจากตัวแปรโครงสร้าง pt ในฟังก์ชัน main() มายังตัวแปร p1 ในฟังก์ชัน printPoint() หลังจากนั้นจึงนำค่าดังกล่าวไปใช้งาน พิจารณาตัวอย่างเพิ่มเติมในตัวอย่างที่ 7.4 ซึ่งเป็นการเขียนตัวอย่างที่ 7.1 ใหม่ในลักษณะของฟังก์ชัน และเพิ่มฟังก์ชันการทำงานต่าง ๆ

## ตัวอย่างที่ 7.4 โปรแกรมตัวอย่างของการใช้ฟังก์ชันในลักษณะต่าง ๆ ซึ่งเกี่ยวข้องกับจุดบนแกนโคออดิเนต

```

#include <stdio.h>
#define MAX_X 200 /* จุดสูงสุดในแนวแกน x ของรูปสี่เหลี่ยม */
#define MAX_Y 300 /* จุดสูงสุดในแนวแกน y ของรูปสี่เหลี่ยม */
#define YES 1
#define NO 0
struct point {
 int x;
 int y;
};
struct rectangle {
 struct point pt1;
 struct point pt2;
};
/* ฟังก์ชันเพื่อใช้ในการกำหนดจุดเริ่มต้นของสี่เหลี่ยมให้กับโครงสร้าง */
struct point makepoint(int x, int y) {
 struct point temp;
 temp.x = x;
 temp.y = y;
 return (temp);
}
/* ฟังก์ชันเพื่อใช้ในการบวกค่า x และ y ของจุด 2 จุด */
struct point addpoint(struct point p1, struct point p2) {
 p1.x += p2.x;
 p1.y += p2.y;
 return (p1);
}

/* ฟังก์ชันเพื่อใช้ตรวจสอบว่าจุดของในรูปสี่เหลี่ยมหรือไม่ */
/* ถ้าไม่อยู่จะคืนค่า 0 ถ้าอยู่จะคืนค่าที่ไม่ใช่ 0 */
int pinrect(struct point p, struct rectangle r) {
 return (p.x >= r.pt1.x && p.x <= r.pt2.x && p.y >= r.pt1.y && p.y <= r.pt2.y);
}

void main() {
 struct rectangle screen;
 struct point middle, sumPoint, specPoint;

```



```

int ans;
screen.pt1 = makepoint(20,30); /* กำหนดจุดแรกของสี่เหลี่ยม */
screen.pt2 = makepoint(MAX_X, MAX_Y); /* กำหนดจุดที่สองของสี่เหลี่ยม */
printf("\nLower bound : %d, %d", screen.pt1.x, screen.pt1.y);
printf("\nUpper bound : %d, %d", screen.pt2.x, screen.pt2.y);

/* หาคู่กึ่งกลางของรูปสี่เหลี่ยม */
middle = makepoint((screen.pt1.x+screen.pt2.x)/2, (screen.pt1.y+screen.pt2.y)/2);
printf("\nMiddle point : %d, %d", middle.x, middle.y);

/* หาผลบวกระหว่างจุดแรกและจุดที่ 2 ของสี่เหลี่ยม */
sumPoint = addpoint(screen.pt1, screen.pt2);
printf("\nSum of bound : %d, %d", sumPoint.x, sumPoint.y);

/* รับค่าจุดและทดสอบว่าจุดนั้นอยู่ในรูปสี่เหลี่ยมที่ระบุหรือไม่ */
printf("\n\nTest point in rectangle");
printf("\nEnter x : ");
scanf("%d", &(specPoint.x));
printf("Enter y : ");
scanf("%d", &(specPoint.y));

ans = pinrect(specPoint, screen);
if (ans == YES)
 printf("\nPoint is in rectangle");
else
 printf("\nPoint is outside rectangle");
}

```

## 5. การใช้พอยน์เตอร์กับตัวแปรโครงสร้าง

กรณีการใช้ฟังก์ชันโดยมีการคืนค่าเป็นตัวแปรโครงสร้าง หรือการส่งอาร์กิวเมนต์เป็นตัวแปรโครงสร้างไปยังฟังก์ชันจะไม่เหมาะกับตัวแปรโครงสร้างที่มีขนาดใหญ่ เนื่องจากทุกครั้งที่ส่งตัวแปรโครงสร้างจะเป็นการจองพื้นที่ตัวแปรใหม่ขึ้นในฟังก์ชัน และมีการสำเนาค่าไปยังตัวแปรตัวใหม่ในฟังก์ชัน ซึ่งจะทำให้ช้าและเปลืองพื้นที่หน่วยความจำ ซึ่งปัญหานี้สามารถให้พอยน์เตอร์เข้ามาช่วยแก้ปัญหา โดยส่งแอดเดรสของตัวแปรโครงสร้างมายังฟังก์ชันที่รับอาร์กิวเมนต์เป็นพอยน์เตอร์ อาร์กิวเมนต์จะชี้ไปยังแอดเดรสเริ่มต้นของตัวแปรโครงสร้างจะช่วยให้การทำงานเร็วขึ้นและเปลืองหน่วยความจำน้อยลง ไม่ว่าโครงสร้างนั้นมีขนาดเท่าใดก็จะใช้พื้นที่และเวลาในการกำหนดแอดเดรสเท่ากัน

เสมอ แต่สิ่งที่ต้องระวังคือหากมีการเปลี่ยนแปลงค่าที่พอยน์เตอร์ที่อยู่ ค่าในตัวแปรโครงสร้างที่ส่งมายังฟังก์ชันจะเปลี่ยนตามโดยอัตโนมัติ การประกาศตัวแปรพอยน์เตอร์ที่ไปยัง struct มีหลักการดังนี้

```
struct point *pp;
```

จะได้ตัวแปรพอยน์เตอร์ pp ที่ไปยังข้อมูลแบบโครงสร้างชื่อ struct point การเขียน \*pp จะเป็นการอ้างถึงโครงสร้าง การอ้างถึงสมาชิกสามารถทำได้โดยอ้าง (\*pp).x หรือ (\*pp).y ดังตัวอย่าง

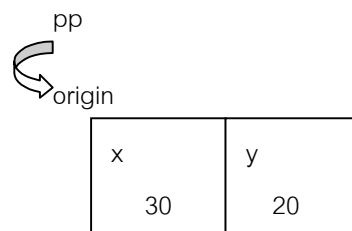
```
struct point origin={30, 20};
```

```
struct point *pp;
```

```
pp = &origin;
```

```
printf ("origin is (%d, %d)\n", (*pp).x, (*pp).y);
```

แสดงสถานะของการทำงานได้ดังรูปที่ 7.6



รูปที่ 7.6 แสดงพอยเตอร์ที่ไปยังตัวแปรโครงสร้าง

สิ่งที่ต้องระวังคือ (\*pp).x จะไม่เหมือนกับ \*pp.x เนื่องจากเครื่องหมาย . จะมีลำดับความสำคัญสูงกว่า \* ทำการแปลความหมาย \*pp.x จะเหมือนกับการอ้าง \*(pp.x) ซึ่งจะก่อให้เกิดความผิดพลาดขึ้น การอ้างถึงสมาชิกในตัวแปรพอยน์เตอร์ที่ไปยังโครงสร้างอาจเขียนอีกลักษณะหนึ่งโดยใช้เครื่องหมาย -> สมมติ p เป็นพอยน์เตอร์ รูปแบบการใช้เป็นดังนี้

```
p->member-of-structure
```

จะสามารถแปลงประโยคการใช้พอยน์เตอร์อ้างสมาชิกของโครงสร้างจากตัวอย่างข้างบนได้ว่า

```
printf ("origin is (%d, %d)\n", pp->x, pp->y);
```

การอ้างถึง (\*pp).x จะมีผลเหมือนกับการอ้างถึง pp->x แสดงตัวอย่างเพิ่มเติมเกี่ยวกับการอ้างถึงสมาชิกในโครงสร้างดังนี้

```
struct rect r, *pr = r;
```

การอ้างถึงสมาชิกต่อไปนี้มีผลเท่ากับการอ้างถึงสมาชิกตัวเดียวกัน

```
r.pt1.x
```

```
(*pr).pt1.x
```

```
pr->pt1.x
```

แสดงตัวอย่างของการใช้พอยน์เตอร์ดังตัวอย่างที่ 7.5

### ตัวอย่างที่ 7.5 แสดงตัวอย่างการอ้างใช้ตัวแปรพอยน์เตอร์กับโครงสร้าง

---

```
#include <stdio.h>
struct point {
 int x;
 int y;
};
struct rectangle {
 struct point pt1;
 struct point pt2;
};
void main() {
 struct point origin={30,20};
 struct point *pp;
 struct rectangle r={{30,20},{100,200}};
 struct rectangle *pr;

 pp = &origin;
 printf("X : %d, Y : %d", (*pp).x, (*pp).y);

 pr = &r;
 printf("\nr.pt1.x = %d", r.pt1.x);
 printf("\n(*pr).pt1.x = %d", (*pr).pt1.x);
 printf("\npr->pt1.x = %d", pr->pt1.x);
}
```

---

การส่งใช้ตัวแปรพอยน์เตอร์ในฟังก์ชัน โดยการส่งแอดเดรสของตัวแปรโครงสร้างไปยังฟังก์ชันนั้น สามารถทำได้เหมือนกับตัวแปรทั่วไป แสดงดังตัวอย่าง

```
struct point pt;
readPoint (&pt);
```

ภายในฟังก์ชัน main( ) มีการเรียกใช้ฟังก์ชัน readPoint( ) โดยส่งแอดเดรสของตัวแปรโครงสร้าง pt ไปยังฟังก์ชัน โปรโตไทป์ของฟังก์ชัน readPoint จะต้องประกาศดังนี้

```
void readPoint (struct point *);
```

การประกาศฟังก์ชัน readPoint() จะทำโดยใช้คำสั่ง

```
void readPoint(struct point *pPt);
```

กระบวนการที่เกิดขึ้น คือ จะมีการจองพื้นที่หน่วยความจำให้กับตัวแปรพอยน์เตอร์ pPt และมีการสำเนาค่าแอดเดรสของตัวแปรโครงสร้าง pt ในฟังก์ชัน main() มาเก็บยัง pPt หรือกล่าวได้ว่าให้ตัวแปร pPt ที่ไปยังตัวแปรโครงสร้าง pt พิจารณาจากการใช้งานจากตัวอย่างที่ 7.6

### ตัวอย่างที่ 7.6 แสดงการใช้งานตัวแปรพอยน์เตอร์โดยปรับปรุงจากตัวอย่างที่ 7.3

---

```
#include <stdio.h>
struct point {
 int x;
 int y;
};
void readPoint(struct point *);
void printPoint(struct point *);
void main() {
 struct point pt;
 readPoint(&pt);
 printPoint(&pt);
}
void readPoint(struct point *pPt) {
 printf("Enter point x : ");
 scanf("%d", &pPt->x); /* scanf("%d", &(*pPt).x); */
 printf("Enter point y : ");
 scanf("%d", &pPt->y); /* scanf("%d", &(*pPt).y); */
}
void printPoint(struct point *pP) {
 printf("\n\nPoint x : %d, y : %d", pP->x, pP->y); /* printf("\n\nPoint x : %d, y : %d", (*pP).x, (*pP).y); */
}

```

---

## 6. การใช้คำสั่ง typedef กับโครงสร้าง

การใช้คำสั่ง typedef ซึ่งเป็นคำสั่งที่ใช้กำหนดชนิดข้อมูลใหม่ มักจะมีการนำมาใช้กับโครงสร้างอยู่เสมอ โดยมีการใช้ใน 2 ลักษณะแสดงดังตัวอย่าง

```

struct point {
 int x;
 int y;
};
typedef struct point Point;
struct point p1;
Point p2;
(1)

```

```

typedef struct {
 int x;
 int y;
} Point;
Point p1, p2;
(2)

```

จากตัวอย่างจะสามารถอ้างถึงชนิดข้อมูลเป็น Point แทนที่จะใช้ struct point ได้ ซึ่งสามารถใช้แทนกันได้ในทุก ๆ กรณี การใช้ typedef ทั้ง 2 ลักษณะมีสิ่งที่แตกต่างกันคือ หากใช้รูปแบบแรกนักเขียนโปรแกรมยังสามารถนำ struct point ไปใช้ได้อีกในโปรแกรม ในขณะที่รูปแบบที่ 2 ไม่สามารถทำได้

พิจารณาตัวอย่างของการทำงาน โดยมีโครงสร้างของวัน ซึ่งประกอบด้วย วันที่ เดือน และปี จะได้ว่า

```

typedef struct {
 int day;
 int month;
 int year;
} Date;

```

สามารถนำไปใช้ประกอบการประกาศตัวแปรได้ เช่น

```
Date today;
```

เป็นการประกาศตัวแปรชื่อ today มีชนิดข้อมูลเป็นโครงสร้าง Date การประกาศตัวแปรตัวชี้ (pointer) ที่ไปยังโครงสร้างก็สามารถทำในลักษณะเดียวกัน เช่น

```
Date *ptrdate; /* 1 */
```

จะได้ตัวแปรพอยน์เตอร์ ptrdate ที่ไปยังโครงสร้าง Date หรืออาจประกาศในลักษณะที่ไม่ใช้ typedef ก็ได้ เช่น

นอกจากนี้เรายังสามารถใช้ typedef ในการกำหนดชื่อชนิดข้อมูลพอยน์เตอร์ที่ไปยังโครงสร้างได้ เช่น

```
typedef Date *PtrDate;
```

```
PtrDate ptrdate; /* 2 */
```

การประกาศตัวแปร ptrdate ทั้ง 2 ลักษณะจะสามารถใช้งานได้เหมือนกันทั้งหมด หากต้องการให้ ptrdate ชี้ไปยังตัวแปรโครงสร้างสามารถทำได้ดังนี้

```
ptrdate = &today;
```

การอ้างถึงสมาชิกของโครงสร้างผ่านตัวแปรตัวชี้ (pointer) สามารถทำได้ดังนี้

```
ptrdate->day = 7;
```

```
if (ptrdate->day == 31 && ptrdate->month == 12)
```

```
scanf ("%d", &ptrdate->year);
```

การอ้างถึงสมาชิกโครงสร้างโดยใช้เครื่องหมาย -> จะมีค่าเท่ากับการใช้คำสั่งต่าง ๆ ดังนี้

```
(*ptrdate).day = 7;
```

```
if ((*ptrdate).day == 31 && (*ptrdate).month == 12)
```

```
scanf ("%d", &((*ptrdate).year));
```

แสดงตัวอย่างการใช้งานดังตัวอย่างที่ 7.7

**ตัวอย่างที่ 7.7** โปรแกรมใช้โครงสร้างของวัน เก็บวันที่ เดือน ปี โดยให้รับข้อมูลวันเดือนปีจากผู้ใช้ หากวันที่ตรงกับวันที่ 1 เดือน 1 ให้พิมพ์ข้อความว่า Happy New Year และตามด้วยปี แต่ถ้าตรงกับวันที่ 25 เดือน 12 ให้ขึ้นข้อความว่า Merry Christmas แต่ถ้าไม่ตรงกับวันที่กำหนดให้ขึ้นข้อความว่า End program

---

```
#include <stdio.h>
```

```
typedef struct {
```

```
int day;
```

```
int month;
```

```
int year;
```

```
} Date;
```

```
void readDate(Date *);
```

```
void printDate(Date *);
```

```
void main() {
```

```
 Date today;
```

```
 readDate(&today);
```

```
 printDate(&today);
```

```
}
```

```

void readDate(Date *d) {
 printf("Enter Date : ");
 scanf("%d", &d->day);
 printf("Enter Month : ");
 scanf("%d", &d->month);
 printf("Enter year : ");
 scanf("%d", &d->year);
}

void printDate(Date *d) {
 if (d->day == 1 && d->month == 1)
 printf("Happy New Year %d", d->year);
 else if (d->day == 25 && d->month == 12)
 printf("Merry Christmas");
 else
 printf("End program");
}

```

## 7. การใช้ตัวแปรชุดเป็นสมาชิกของโครงสร้าง

สมาชิกภายในตัวแปรชุดสามารถเป็นข้อมูลประเภทใดก็ได้ ทั้งข้อมูลพื้นฐานและข้อมูลประเภทอื่น การใช้ตัวแปรชุดเป็นสมาชิกของโครงสร้าง สามารถทำได้ดังนี้ สมมติให้สร้างโครงสร้างเพื่อเก็บข้อมูลของนักเรียนคนหนึ่ง ซึ่งประกอบด้วย ชื่อ นามสกุล คะแนนสอบครั้งที่ 1 2 และ 3 และคะแนนรวม จะสามารถประกาศโครงสร้างได้ดังนี้

```

typedef struct {
 char name[16];
 char surname[20];
 float score[3];
 float total;
} Student ;

```

ข้อมูลเรื่องของคะแนนสอบทั้ง 3 ครั้งจะเก็บอยู่ที่ score ในลักษณะตัวแปรชุด ส่วนข้อมูล name และ surname หากพิจารณาเป็นข้อมูลเดียวกันก็เป็นข้อมูลสตริง แต่ก็สามารถพิจารณาใช้งานในลักษณะของตัวแปรชุดของตัวอักขระได้เช่นเดียวกัน สามารถค่าเริ่มต้นให้กับโครงสร้างดังกล่าว และใช้งานโครงสร้างในลักษณะเดียวกับโครงสร้างและตัวแปรชุดทั่วไป เช่น

```

Student std={"Somchai", "Jaidee", 10.0, 20.0, 30.0};
printf("%s", std.surname); /* อ้างถึงข้อมูลนามสกุลทั้งหมด เป็นสตริง */

```

```
printf("%c", std.name[0]); /* อ้างถึงตัวอักษรตัวแรกของชื่อ เป็น char */
printf("%.2f", std.score[0]); /* อ้างถึงคะแนนสอบครั้งที่ 1 เก็บในตัวแปรชุด */
```

แสดงการใช้งานดังตัวอย่างที่ 7.8

**ตัวอย่างที่ 7.8** โปรแกรมเพื่อเก็บข้อมูลนักเรียนคนหนึ่ง ประกอบด้วย ชื่อ นามสกุล คะแนนสอบแต่ละครั้ง ตั้งแต่ครั้งที่ 1 ถึง 5 ซึ่งแบ่งออกเป็นครั้งละ 20 คะแนนเท่า ๆ กัน และคะแนนรวมของนักเรียนคนนั้น โปรแกรมจะมีความสามารถต่าง ๆ ดังนี้

- รับข้อมูลชื่อนามสกุล และคะแนนสอบแต่ละครั้งของนักเรียน
- หาคะแนนรวมของนักเรียนคนนั้น
- หาคะแนนเฉลี่ยของการสอบแต่ละครั้ง
- หาว่ามีการสอบครั้งใดที่นักเรียนคนนั้นได้คะแนนน้อยกว่า 10 คะแนน และได้คะแนนเท่าใด

---

```
#include <stdio.h>
#define NO_SCORE 5
typedef struct {
 char name[16];
 char surname[20];
 int score[NO_SCORE];
 float total;
} Student;
void readStudentData(Student *);
void findTotalScore(Student *);
float findAverage(Student);
void findLessThanTen(Student);
void main() {
 Student std;
 float avg;
 readStudentData(&std);
 findTotalScore(&std);
 avg = findAverage(std);
 printf("\n\nAverage score is %.2f", avg);
 findLessThanTen(std);
}
void readStudentData(Student *pStd) {
 int i;
 printf("Enter student data\n");
```



```

printf("\tName : ");
scanf("%s", &pStd->name);
printf("\tSurname : ");
scanf("%s", &pStd->surname);
for (i=0; i<NO_SCORE; i++) {
 printf("\tScore %d : ", i+1);
 scanf("%d", &pStd->score[i]);
}
}
void findTotalScore(Student *pStd) {
 int i;
 printf("\n\nPrint student data");
 printf("\n\t%s %s got score ", pStd->name, pStd->surname);
 pStd->total = 0.0;
 for (i=0; i<NO_SCORE; i++) {
 printf("%6d", pStd->score[i]);
 pStd->total += pStd->score[i];
 }
 printf("\n\tTotal score is %.2f", pStd->total);
}
float findAverage(Student s) {
 return(s.total/NO_SCORE);
}
void findLessThanTen(Student s) {
 int i,count=0;
 printf("\n\nScore less than 10");
 for (i=0; i<NO_SCORE; i++) {
 if (s.score[i] < 10) {
 printf("\n\tTest no.%d - %d", i+1, s.score[i]);
 count++;
 }
 }
 if (count==0) /* กรณีที่ไม่มีการสอบครั้งใดได้น้อยกว่า 10 */
 printf(" -> None");
}

```

---

**หมายเหตุ** จากการใช้ Turbo C เวอร์ชัน 3.0 เป็นคอมไพเลอร์ในบางเครื่องจะพบว่า หากภายในโครงสร้างมีข้อมูลที่เป็นจำนวนจริง (float หรือ float) และมีการรับข้อมูลสมาชิกตัวนั้น จะเกิดความผิดพลาดในช่วงของการรัน

โปรแกรม แต่โปรแกรมนั้นสามารถนำไปคอมไพล์และรันโดยคอมไพเลอร์อื่นได้ เช่น gcc บนระบบปฏิบัติการลินุกซ์ (Linux) และซันโซลาริส (Sun Solaris) ในที่นี้จึงกำหนดให้ข้อมูล score เป็น int เพื่อให้สามารถทำงานกับ Turbo C ได้

## 8. ตัวแปรชุดของโครงสร้าง

การใช้งานตัวแปรโครงสร้างนอกจากใช้ในลักษณะของตัวแปรเดี่ยวแล้ว ยังสามารถใช้งานตัวแปรโครงสร้างในลักษณะของตัวแปรชุดได้อีกด้วย ซึ่งเป็นเรื่องสำคัญและใช้บ่อยเมื่อมีการเขียนโปรแกรมเพื่อใช้งานธุรกิจ เช่น การเก็บข้อมูลประวัติของพนักงาน จะมีโครงสร้างที่ใช้เก็บข้อมูลของพนักงานแต่ละคน หากใช้ในลักษณะของตัวแปรปกติ จะสามารถเก็บข้อมูลของพนักงานได้เพียง 1 คน ซึ่งพนักงานทั้งบริษัทอาจจะมีหลายสิบหรือหลายร้อยคน การเก็บข้อมูลในลักษณะนี้จะใช้ตัวแปรชุดเข้ามาช่วย เพราะช่วยให้การบำรุงรักษาโปรแกรมทำได้ง่ายขึ้น แต่หากข้อมูลมีปริมาณมากและมีจำนวนสมาชิกของตัวแปรชุดที่ไม่แน่นอน มักจะใช้โครงสร้างข้อมูลประเภทลิสต์ (List) เข้ามาช่วยเนื่องจากช่วยประหยัดเนื้อที่หน่วยความจำได้มากกว่า ซึ่งหารายละเอียดได้ในหนังสือการเขียนโปรแกรมขั้นสูง และหนังสือโครงสร้างข้อมูลทั่วไป ตัวอย่างของการใช้ตัวแปรชุดของโครงสร้าง ได้แก่

```
#define STAFFSIZE 100
typedef struct {
 char employee_id[4];
 char name[16];
 char surname[20];
 char gender; /* 0 - Male, 1 - Female */
 char department[10];
 float salary;
} Employee ;
Employee staff[STAFFSIZE];
```

การอ้างโดยใช้คำสั่งต่าง ๆ

|                  |                                              |
|------------------|----------------------------------------------|
| staff            | อ้างถึงตัวแปรชุดของโครงสร้าง                 |
| staff[i]         | อ้างถึงสมาชิกตัวที่ i ในตัวแปรชุด            |
| staff[i].name[i] | อ้างถึงตัวอักษรตัวแรกในชื่อของพนักงานคนที่ i |
| staff[i].surname | อ้างถึงนามสกุลของพนักงานคนที่ i              |

หากต้องการเรียกใช้งานฟังก์ชันที่ทำงานกับตัวแปรชุด เช่น ฟังก์ชันการพิมพ์ชื่อพนักงานทั้งบริษัท

```
printEmployee (staff) ;
```

รูปแบบฟังก์ชันสามารถกำหนดด้วย

```
void printEmployee (Employee emp [])
```

การส่งตัวแปรชุดไปในฟังก์ชันสามารถอ้างด้วยชื่อตัวแปรชุดได้ทันที การทำงานเป็นลักษณะเดียวกับพอยน์เตอร์ การเปลี่ยนแปลงค่าในฟังก์ชันจะเป็นการเปลี่ยนแปลงค่าตัวแปรชุดในฝั่งที่เรียกใช้ฟังก์ชันด้วย เช่น ฟังก์ชันของการอ่านชื่อพนักงาน ก็จะส่งในลักษณะเดียวกับฟังก์ชัน printEmployee( )

หากต้องการเรียกใช้ฟังก์ชันที่มีการอ้างถึงสมาชิกแต่ละตัวในตัวแปรชุด ลักษณะการใช้งานจะเหมือนกับการใช้งานตัวแปรโครงสร้าง แต่ใช้ระบบดัชนีมาอ้างถึงสมาชิกเหมือนกับตัวแปรชุดทั่วไป เช่น ฟังก์ชันที่ใช้ในการพิมพ์ชื่อสมาชิกคนที่ระบุ จะเรียกใช้โดย

```
printPerson (staff[k]);
```

รูปแบบฟังก์ชันสามารถกำหนดด้วย

```
void printPerson (Employee person)
```

แต่หากต้องการอ่านข้อมูลของพนักงานทีละคน โดยเรียกใช้ในลักษณะฟังก์ชัน ก็สามารถใช้ในลักษณะของฟังก์ชันคืนค่าเป็นโครงสร้าง หรือส่งอาร์กิวเมนต์เป็นพอยน์เตอร์ให้กับฟังก์ชันก็ได้ เช่น

```
staff[k] = readPerson(); ใช้คู่กับฟังก์ชัน Employee readPerson() { ... }
```

หรือใช้ในลักษณะส่งพอยน์เตอร์ไปยังฟังก์ชันได้ดังคำสั่ง

```
readPerson (&staff[k]); ใช้คู่กับฟังก์ชัน void readPerson(Employee *person) { ... }
```

การกำหนดค่าเริ่มต้นให้กับตัวแปรชุดของโครงสร้างสามารถทำได้โดย

```
Person staff[] = { { "0001", "Somchai", "Jaidee", 0, "Account", 10000.0 },
 { "0002", "Somsak", "Tamdee", 0, "Sale", 8000.0 },
 { "0003", "Somsri", "Deejai", 1, "Customer", 25 } };
```

หากกำหนดในลักษณะดังกล่าวจะได้ว่าตัวแปรชุดมีขนาดสมาชิกคือ 3 ขอบเขตข้อมูลแต่ละสมาชิกจะอยู่ภายในเครื่องหมาย {} ตัวอย่างเพิ่มเติมของการใช้งานตัวแปรชุดของโครงสร้างแสดงดังตัวอย่างที่ 7.9 และ 7.10

**ตัวอย่างที่ 7.9** โปรแกรมเพื่อรับข้อมูลชื่อ นามสกุล และอายุของนักเรียนห้องหนึ่งซึ่งมี 20 คน โปรแกรมสามารถหาอายุเฉลี่ยของนักเรียนทั้งห้อง หาว่ามีนักเรียนคนใดที่มีอายุน้อยกว่าอายุเฉลี่ย และหาชื่อนักเรียนที่มีอายุมากที่สุดและน้อยที่สุดของนักเรียนในห้องนั้น

---

```
#include <stdio.h>
#define MAX_STUDENT 4
typedef struct {
 char name[16];
 char surname[20];
 int age;
} Student;
void readStudent(Student []);
```

```

void printStudent(Student []);
float findAverageAge(Student []);
void LessThanAverage(Student [], float);
void findMaxMinAge(Student [], int *, int *);
void printMaxAge(Student [], int);
void printMinAge(Student [], int);
void main() {
 Student student[MAX_STUDENT];
 float avg;
 int maxAge, minAge;
 readStudent(student);
 printStudent(student);
 avg = findAverageAge(student);
 printf("\n\nAverage age is %.2f", avg);
 LessThanAverage(student, avg);
 findMaxMinAge(student, &maxAge, &minAge);
 printMaxAge(student, maxAge);
 printMinAge(student, minAge);
}
void readStudent(Student stdarr[]) {
 int i;
 printf("Enter student data");
 for (i=0; i<MAX_STUDENT; i++) {
 printf("\n\tNo.%d", i+1);
 printf("\n\tName : ");
 scanf("%s", stdarr[i].name);
 printf("\tSurname : ");
 scanf("%s", stdarr[i].surname);
 printf("\tAge : ");
 scanf("%d", &stdarr[i].age);
 }
}
void printStudent(Student stdarr[]) {
 int i;
 printf("\n\nStudent data");
 for (i=0; i<MAX_STUDENT; i++) {
 printf("\n\tNo.%d", i+1);
 printf("\tName : %16s", stdarr[i].name);
 }
}

```

```

 printf("\tSurname : %20s", stdarr[i].surname);
 printf("\tAge : %d", stdarr[i].age);
}
}

float findAverageAge(Student stdarr[]) {
 float sumAge=0.0, avgAge;
 int i;
 for (i=0; i<MAX_STUDENT; i++) {
 sumAge = sumAge + stdarr[i].age;
 }
 avgAge = sumAge / MAX_STUDENT;
 return(avgAge);
}

void LessThanAverage(Student stdarr[], float avg) {
 int i;
 printf("\n\nLess than average age %.2f", avg);
 for (i=0; i<MAX_STUDENT; i++) {
 if (stdarr[i].age < avg)
 printf("\n%s %s with %d years old", stdarr[i].name, stdarr[i].surname, stdarr[i].age);
 }
}

void findMaxMinAge(Student stdarr[], int *max, int *min) {
 int i, maxIdx, minIdx;
 maxIdx = minIdx = 0;
 for (i=1; i<MAX_STUDENT; i++) {
 if (stdarr[i].age > stdarr[maxIdx].age)
 maxIdx = i;
 else if (stdarr[i].age < stdarr[minIdx].age)
 minIdx = i;
 }
 *max = stdarr[maxIdx].age;
 *min = stdarr[minIdx].age;
}

void printMaxAge(Student stdarr[], int maxAge) {
 int i;
 printf("\n\nMaximum age report");
 printf("\n\nMaximum age is %d", maxAge);
 for (i=0; i<MAX_STUDENT; i++) {

```

```

 if (stdarr[i].age == maxAge)
 printf("\n\t%s %s", stdarr[i].name, stdarr[i].surname);
 }
}

void printMinAge(Student stdarr[], int minAge) {
 int i;
 printf("\n\nMinimum age report");
 printf("\nMinimum age is %d", minAge);
 for (i=0; i<MAX_STUDENT; i++) {
 if (stdarr[i].age == minAge)
 printf("\n\t%s %s", stdarr[i].name, stdarr[i].surname);
 }
}
}

```

---

**ตัวอย่างที่ 7.10** โปรแกรมคำนวณอายุบนดวงดาวอื่น โดยใช้โครงสร้างและชนิดข้อมูลอื่น ๆ ในการคำนวณอายุ หากอายุคน 18 ปี แสดงว่าเป็นการหมุนของโลกรอบดวงอาทิตย์ผ่านไป 18 รอบ เราสามารถคำนวณอายุของคนบนดาวดวงอื่นโดยอาศัยการคำนวณด้วยสูตร

$$X = (Y * 365) / D$$

เมื่อ X แทนอายุบนดาวที่ระยะ Y แทนอายุบนโลก D แทนจำนวนวันที่ดาวที่ระยะโคจรรอบดวงอาทิตย์ ให้สร้างโครงสร้างเพื่อเก็บข้อมูลชื่อผู้ใช้ อายุบนโลก ชื่อดาวที่ระยะ และอายุบนดาวที่ระยะ นอกจากนี้ให้สร้างโครงสร้างในลักษณะตัวแปรชุดเพื่อเก็บข้อมูลการโคจรรอบดาวต่าง ๆ และกำหนดค่าเริ่มต้นให้กับตัวแปรของโครงสร้างดังกล่าว ดังข้อมูลต่อไปนี้

| ชื่อดาว | จำนวนวันที่โคจรรอบดวงอาทิตย์ |
|---------|------------------------------|
| Mercury | 88                           |
| Venus   | 225                          |
| Jupiter | 4380                         |
| Saturn  | 10767                        |

```

#include <stdio.h>
#include <conio.h>
#include <string.h>

```

```

typedef struct {
 char name[20];

```

```

int earthAge;
char targetName[20];
int targetAge;
} ManInfo;

typedef struct {
 char name[20];
 int sunOrbit;
} PlanetInfo;

void readManInfo(ManInfo *);
void printManInfo(ManInfo);
int selectPlanet(PlanetInfo []);
void computeAge(ManInfo *, PlanetInfo [], int);
void main() {
 PlanetInfo targetPlanet[4]={"Mercury", 88}, {"Venus", 225}, {"Jupiter",4380}, {"Saturn", 10767}};
 ManInfo man;
 int choice;
 printf("\nProgram to calculate man's age on other planet");
 readManInfo(&man);
 choice = selectPlanet(targetPlanet);
 computeAge(&man, targetPlanet, choice);
 printManInfo(man);
}

void computeAge(ManInfo *pMan, PlanetInfo target[], int ch) {
 int tAge;
 strcpy(pMan->targetName, target[ch].name);
 pMan->targetAge = (pMan->earthAge * 365) / target[ch].sunOrbit;
}

int selectPlanet(PlanetInfo target[]) {
 int sel;
 do {
 printf("\nSelect 0. Mercury, 1. Venus, 2. Jupiter, 3. Saturn : ");
 scanf("%d", &sel);
 } while (sel < 0 || sel > 3);
 return(sel);
}

void printManInfo(ManInfo tMan) {
 printf("\n\nResult");

```

```

printf("\nName : %s", tMan.name);
printf("\nAge on Earth : %d", tMan.earthAge);
printf("\nAge on %s : %d", tMan.targetName, tMan.targetAge);
}

void readManInfo(ManInfo *pMan) {
printf("\nEnter your name : ");
scanf("%s", pMan->name);
printf("Enter your age : ");
scanf("%d", &pMan->earthAge);
}

```

---

## 9. ยูเนียน (Union)

ยูเนียนเป็นชนิดข้อมูลที่คล้ายกับชนิดข้อมูลโครงสร้าง คือ เป็นชนิดข้อมูลที่ประกอบด้วยสมาชิกที่มีชนิดข้อมูลหลาย ๆ ประเภท การใช้งานทุกต่าง ๆ จะใช้ในลักษณะเดียวกับข้อมูลโครงสร้าง ทั้งการอ้างถึงสมาชิก การส่งยูเนียนให้กับฟังก์ชัน และอื่น ๆ แต่ต่างกันที่ในขณะที่ใดขณะหนึ่งนั้นยูเนียนจะสามารถใช้สมาชิกได้เพียงตัวเดียวเท่านั้น ยูเนียนจะช่วยให้การจัดการข้อมูลที่ต่างชนิดกันในพื้นที่ของตัวแปรยูเนียนเดียวกัน เช่น

```

union number { /*members are overlaid */
 int integer;
 float decimal;
};

typedef union number Number;

Number data;

```

การจองพื้นที่ของตัวแปร data จะจองให้กับสมาชิกของยูเนียนทั้ง 2 ตัว แต่ในขณะที่ใดขณะหนึ่งจะสามารถใช้สมาชิกได้แค่ตัวเดียวเท่านั้น ซึ่งผู้เขียนโปรแกรมจะต้องเป็นผู้ควบคุมการใช้ด้วยตัวเอง คอมไพเลอร์จะไม่ตรวจสอบความผิดพลาดให้ แสดงตัวอย่างการใช้งานดังตัวอย่างที่ 7.11

### ตัวอย่างที่ 7.11 ตัวอย่างการใช้สมาชิกของยูเนียน

```

#include <stdio.h>

union number {
 int integer;
 float decimal;
};

```



```

typedef union number Number;

void main () {
 Number data;
 data.integer = 20000;
 printf ("int : %6d, float : %10.4f\n", data.integer, data.decimal);
 data.decimal = 123.0;
 printf ("int : %6d, float : %10.4f\n", data.integer, data.decimal);
}

```

---

ผลลัพธ์ของการทำงานจะขึ้นอยู่กับการทำงานแต่ละครั้ง แต่การใช้ฟังก์ชัน printf ครั้งแรกจะให้คำตอบจำนวนเต็มที่ถูกต้อง ในขณะที่จำนวนจริงจะได้ค่าที่ไม่สามารถคาดเดาได้ และการใช้ฟังก์ชัน printf ครั้งที่ 2 จะให้คำตอบจำนวนจริงที่ถูกต้อง ในขณะที่จำนวนเต็มจะได้ค่าที่ไม่สามารถคาดเดาได้ เช่น

```
int : 20000, float : 0.0000
```

```
int : 31553, float : 123.0000
```

การอ้างถึงสมาชิกภายในยูเนียนสามารถใช้

```
union-variable.member-name
```

หรือ

```
union-variable->member-name
```

แสดงตัวอย่างการใช้งานดังตัวอย่างที่ 7.12

### ตัวอย่างที่ 7.12 โปรแกรมแสดงการใช้งานยูเนียน

---

```

#include <stdio.h>
typedef enum { INT, FLOAT } Tag;
union number { /* number pair template */
 int integer;
 float decimal;
};
typedef union number Number;
struct pair { /* tagged number pairs */
 Tag tagged;
 Number value;
};
typedef struct pair Pair;
void print_pair (Pair);

```

```

main () {
 Pair data;
 data.tagged = INT;
 data.value.integer = 20000;
 print_pair (data);
 data.tagged = FLOAT;
 data.value.decimal = 123.0;
 print_pair (data);
}

void print_pair (Pair d) {
 switch (d.tagged) {
 case INT : printf ("Integer : %d\n", d.value.integer);
 break;
 case FLOAT : printf ("Decimal : %f\n", d.value.decimal);
 break;
 }
}

```

จากตัวอย่างมีการใช้คำสั่ง enum เป็นคำสั่งช่วยให้ผู้ใช้สามารถกำหนดชนิดข้อมูลใหม่ขึ้นได้เอง เช่น

```
enum workday { Monday, Tuesday, Wednesday, Thursday, Friday };
```

เป็นการกำหนดชนิดข้อมูลชื่อ Workday ประเภทข้อมูลที่เป็นไปได้คือข้อมูลที่อยู่ในเครื่องหมายปีกกาทั้งหมด เมื่อต้องการใช้งานต้องประกาศตัวแปรดังนี้

```
enum workday today;
```

```
today = Monday;
```

```
if (today == Monday)
```

```
 printf("Today is %d", today);
```

ผลลัพธ์ที่ได้จากการทำงานคือ Today is 0 เนื่องจากการเก็บข้อมูลของ enum จะเก็บในลักษณะคล้ายตัวแปรชุดของจำนวนเต็ม โดยที่ข้อมูลตัวแรกมีค่าเป็น 0 และไล่ค่าเป็น 1 2 3 ไปจนกระทั่งหมดข้อมูล และสามารถใช้ enum ร่วมกับคำสั่ง typedef ได้ดังตัวอย่างที่ 7.12

ในการใช้งานยูเนียนมักจะใช้ในลักษณะที่มีตัวควบคุมการทำงานขณะนั้นว่ากำลังใช้งานที่สมาชิกตัวใดของยูเนียน เพื่อให้สามารถเรียกใช้ข้อมูลได้ถูกต้องดังตัวอย่างข้างต้น และมักจะใช้ยูเนียนร่วมกับข้อมูลโครงสร้างเสมอ เช่น ต้องการเก็บข้อมูลผลคะแนนของนักเรียน ซึ่งแบ่งการคิดเกรดเป็น 2 ลักษณะ นักเรียนประเภท A ให้เก็บผลการเรียนเป็นลักษณะร้อยละ แต่ถ้าเป็นนักเรียนประเภท B ให้เก็บข้อมูลในลักษณะเกรด A ถึง F ประกาศโครงสร้างได้ดังนี้

```
typedef enum { A, B } Type;
typedef union {
 float percent;
 char grade;
} Result;
```

```
typedef struct {
 char name[16];
 float score;
 Type type;
 Result result;
} Student ;
```

## แบบฝึกหัดบทที่ 6

1. ใ้หาที่ผิดของคำสั่งต่อไปนี้และแก้ไขให้ถูกต้อง

```
(ก) struct {
 char name[20]
 int age
}
```

```
(ข) struct point {
 int x;
 int y;
};
point p1;
printf("np1 = %d", p1);
```

2. ออกแบบประเภทข้อมูลแบบโครงสร้าง เพื่อเก็บข้อมูลประวัติของนักศึกษาแต่ละคน โดยเก็บ รหัสนักศึกษา ชื่อ ที่อยู่ ชื่อผู้ปกครอง วันเดือนปีเกิด เงินเดือนที่ได้รับ
3. เขียนโปรแกรมเพื่อแสดงผลข้อมูลดังต่อไปนี้

```
ชื่อ
รุ่น
ประเภท
จำนวนประตู
```

กำหนดให้เก็บข้อมูลแบบโครงสร้างและมีฟังก์ชันในโปรแกรมอย่างน้อย 3 ฟังก์ชันรวมทั้งฟังก์ชัน main() โดยที่ฟังก์ชันย่อยทำหน้าที่รับข้อมูล และแสดงผลข้อมูล

4. เขียนโปรแกรมเพื่อคำนวณพื้นที่ของรูปสี่เหลี่ยม โดยเก็บข้อมูลของสี่เหลี่ยมคือ ความกว้าง ความสูง และพื้นที่ เป็นลักษณะโครงสร้าง
5. เขียนโปรแกรมเพื่อทำหน้าที่ในการตัดเกรดของกระบวนวิชาหนึ่ง ซึ่งมีนักศึกษาไม่เกิน 100 คน หากมีการป้อนข้อมูลคะแนนติดลบจะถือเป็นการหยุดการป้อนข้อมูล โดยรับข้อมูลรหัสนักศึกษา และคะแนนสอบของนักศึกษา เก็บไว้ในโครงสร้าง โปรแกรมจะทำหน้าที่ตัดเกรดและแสดงผลการตัดเกรดให้กับนักศึกษาแต่ละคนโดยคิดจากคะแนนเต็ม 100 คะแนน มีเกณฑ์การตัดเกรดดังนี้

| คะแนน             | เกรด |
|-------------------|------|
| ตั้งแต่ 85 ขึ้นไป | A    |
| 75 ถึง 84         | B    |
| 60 ถึง 75         | C    |
| 50 ถึง 59         | D    |
| ต่ำกว่า 50        | F    |

6. เขียนโปรแกรมเพื่อคำนวณรายได้ของพนักงานขายในบริษัทแห่งหนึ่ง หากรหัสพนักงานเป็น 0000 แสดงว่าสิ้นสุดการป้อนข้อมูล ให้รับข้อมูลรหัสพนักงาน ยอดขาย ต้นทุน และให้แสดงรายงานต่าง ๆ คือ รหัสพนักงาน ยอดขาย ต้นทุน กำไร (= ยอดขาย - ต้นทุน) และค่าคอมมิชชัน แสดงดังตัวอย่างข้างล่าง กำหนดว่าหากยอดขายมีมูลค่าไม่เกิน 5000 ให้คิดค่าคอมมิชชัน 3.5 เปอร์เซ็นต์ของกำไร แต่หากเกินกว่านั้นให้คิดคอมมิชชันที่ 5 เปอร์เซ็นต์ของกำไร ตัวอย่างผลลัพธ์การทำงาน

| Number | Sales     | Value    | Profit   | Commission |
|--------|-----------|----------|----------|------------|
| 1234   | 10234.50  | 8750.00  | 1484.50  | 51.96      |
| 2123   | 105400.00 | 85000.00 | 20400.00 | 1020.00    |

7. เขียนโปรแกรมเพื่อเก็บข้อมูลรถยนต์จำนวน 10 รุ่น โดยที่เก็บข้อมูลยี่ห้อ รุ่น ปีที่ผลิต ความเร็วสูงสุด โปรแกรมจะทำหน้าที่
- จัดพิมพ์รายละเอียดของรถยนต์ที่จัดเก็บทั้งหมด
  - ทหารายละเอียดรถยนต์ที่มีความเร็วสูงสุด กรณีที่มีรถยนต์ที่มีความเร็วสูงสุดเท่ากัน ให้พิมพ์รายละเอียดของรถยนต์นั้นทั้งหมด
  - รับข้อมูลปีที่ผลิตจากผู้ใช้ และแสดงผลข้อมูลของรถยนต์ที่ผลิตในปีนั้น
8. เขียนโปรแกรมเพื่อคำนวณค่าเฉลี่ยของคะแนนสอบของนักศึกษา แยกตามประเภทของนักศึกษา คือ นักศึกษาสังกัดคณะวิทยาศาสตร์ และนักศึกษาคณะอื่นๆ หากผู้ใช้ป้อนค่า '1' ให้ถือว่าเป็นนักศึกษาคณะวิทยาศาสตร์ หากป้อนค่าอื่นๆ ให้ถือว่าเป็นนักศึกษาคณะอื่น โดยก่อนการทำงานของโปรแกรมให้สอบถามจำนวนนักศึกษาจากผู้ใช้โดยมีจำนวนนักศึกษาไม่เกิน 80 คน กำหนดให้เก็บข้อมูลสังกัดของนักศึกษาและคะแนนสอบไว้ในตัวแปรชุดของโครงสร้างให้เรียบร้อยก่อน แล้วจึงหาค่าเฉลี่ยของนักศึกษาสังกัดคณะวิทยาศาสตร์ และนักศึกษาสังกัดคณะอื่น
9. เขียนโปรแกรมในลักษณะโครงสร้างและยูนิเอน เพื่อเก็บข้อมูลเงินรายได้ของพนักงานในบริษัทแห่งหนึ่ง ซึ่งมีพนักงานไม่เกิน 50 คน หากมีการป้อนข้อมูลเงินเดือนของพนักงานน้อยกว่า 4,000 บาทจะเป็นการสิ้นสุดการป้อนข้อมูล บริษัทแห่งนี้แบ่งพนักงานเป็น 2 ประเภท คือ พนักงานทั่วไป และพนักงานฝ่ายขาย หากเป็นพนักงานทั่วไปรายได้พนักงานจะได้จากเงินเดือน และโบนัสซึ่งคิดเป็นจำนวนเดือน แต่หากเป็นพนักงานฝ่ายขายจะมีรายได้จากเงินเดือน และค่าคอมมิชชัน และแสดงผลเงินรายได้ของพนักงานทั้งหมดโดยแยกแสดงผลตามประเภทของพนักงาน

10. เขียนโปรแกรมเพื่อสร้างตารางแจกแจงความถี่ของจำนวนนักศึกษาโดยแยกตามคณะและชั้นปีที่นักศึกษาสังกัด แสดงดังตาราง

|           | First Year | Second Year | Third Year | Fourth Year | >4  | Total |
|-----------|------------|-------------|------------|-------------|-----|-------|
| Faculty 1 | 1500       | 1200        | 1150       | 1100        | 20  | 4970  |
| Faculty 2 | 3500       | 3300        | 3200       | 3100        | 100 | 12200 |
| ...       |            |             |            |             |     |       |
| Total     | ...        | ...         | ...        | ...         | ... | ...   |

โปรแกรมจะรับข้อมูลนักศึกษาแต่ละคน เก็บเป็นโครงสร้างประกอบด้วย รหัสนักศึกษา คณะที่สังกัด และชั้นปี หลังจากนั้นให้สร้างตารางแจกแจงความถี่ โดยที่มีการสรุปยอดรวมของนักศึกษาแยกตามคณะ และชั้นปี รวมทั้งสรุปยอดนักศึกษารวมด้วย

11. เขียนโปรแกรมเพื่อทำหน้าที่แปลงค่าเงินจากเงินสกุลอื่นเป็นเงินบาท และจากเงินบาทเป็นเงินสกุลอื่น ให้เก็บข้อมูลสกุลเงิน วิธีการแลกเปลี่ยน จำนวนเงินที่แลกเปลี่ยน จำนวนเงินที่ได้รับ เก็บไว้ในโครงสร้าง และมีข้อมูลอัตราแลกเปลี่ยนเก็บไว้ในตัวแปรชุดของโครงสร้างมีข้อมูลอัตราแลกเปลี่ยนดังนี้

| Foreign   | Exchange Rate (to Baht) |
|-----------|-------------------------|
| 1. Yen    | 37.66 Baht / 100 Yen    |
| 2. Dollar | 45.40 Baht / 1 Dollar   |
| 3. Pound  | 61.838 Baht / 1 Pound   |

การอ้างถึงข้อมูลใด ๆ ที่เกี่ยวข้องให้อ้างถึงผ่านข้อมูลโครงสร้าง และตัวแปรชุดของโครงสร้างที่กำหนดเท่านั้น โดยโปรแกรมจะแสดงตารางอัตราแลกเปลี่ยน เพื่อให้ผู้ใช้เลือกสกุลเงินที่ต้องการแลกเปลี่ยน และให้ผู้ใช้กำหนดวิธีการแลกเปลี่ยน โดย 0 แทนเปลี่ยนเงินสกุลอื่นเป็นเงินบาท และ 1 แทนเปลี่ยนเงินบาทเป็นเงินสกุลเงิน หลังจากนั้นจะให้ผู้ใช้ระบุยอดเงินที่ต้องการเปลี่ยน โปรแกรมจะทำหน้าที่คำนวณและแสดงผลยอดเงินที่ถูกคำนวณได้รับ

กำหนดโปรโตไทป์ของฟังก์ชันที่ทำหน้าที่แปลงค่าเงินดังนี้

- แปลงค่าเงินต่างประเทศเป็นเงินบาท กำหนดให้ใช้วิธีคืนค่าเป็นโครงสร้าง
- แปลงเงินบาทเป็นเงินสกุลอื่น โดยที่ float ตัวแรกเป็นค่าเงินบาท float ตัวที่ 2 เป็นเงินต่างประเทศที่แลกเปลี่ยนได้ กำหนดให้คืนค่าเป็นพอยน์เตอร์

# การจัดการเพิ่มข้อมูล ( File Manipulation )

## 8

เพิ่มข้อมูล หมายถึง กลุ่มของระเบียน (Record) ตั้งแต่ 1 ระเบียนขึ้นไปมารวมกันเป็นเรื่องเดียวกัน ตัวอย่างเช่น รายละเอียดข้อมูลประวัติของนักศึกษาแต่ละคน เรียกว่า ระเบียน แต่เมื่อเอาระเบียนข้อมูลประวัตินักศึกษาเหล่านี้มาเก็บรวมกันจะเรียกว่า เพิ่มข้อมูล

### 1. ฟังก์ชันที่ใช้สำหรับการประมวลผลเพิ่มข้อมูล

ในคลังชุดคำสั่งมาตรฐานของภาษา C ไม่มีคำสั่งที่ใช้สำหรับประมวลผลเพิ่มข้อมูล ดังนั้นในการทำงานกับเพิ่มข้อมูลจึงจำเป็นต้องเรียกใช้แฟ้ม `stdio.h` ซึ่งเป็นแฟ้มที่เก็บคำสั่งและคำจำกัดความต่างๆ ที่จำเป็นต้องใช้ในการประมวลผลเพิ่มข้อมูล คำสั่งที่สำคัญสำหรับการประมวลผลเพิ่มข้อมูลมีดังนี้

#### 1.1 ฟังก์ชัน `fopen()`

ในการใช้งานเพิ่มข้อมูลเพื่อการอ่านหรือการเขียนเพิ่มข้อมูล สิ่งแรกที่ต้องทำคือการเปิดเพิ่มข้อมูลนั้นก่อน โดยต้องดำเนินการตามขั้นตอนดังนี้

```
FILE *fileptr;
fileptr = fopen(filename,mode)
```

ในขั้นตอนแรกเราต้องกำหนดตัวแปรพอยน์เตอร์ไปยังเพิ่มข้อมูล ในที่นี้ให้ชื่อตัวแปรนี้ว่า `fileptr` จากนั้นทำการเรียกฟังก์ชัน `fopen` เพื่อทำการเปิดเพิ่มข้อมูล โดยที่

`filename` คือ ตัวแปรที่เก็บชื่อเพิ่มข้อมูลที่ต้องการเปิด

`mode` คือ รูปแบบของการเปิดเพิ่มข้อมูล

- “w”      ทำการสร้างเพิ่มข้อมูลใหม่และเปิดเพิ่มข้อมูลเพื่อเขียน หากมีเพิ่มข้อมูลนั้นอยู่แล้ว จะทำการลบข้อมูลเดิม และสร้างเพิ่มข้อมูลใหม่
- “r”      ทำการเปิดเพิ่มข้อมูลเพื่อทำการอ่านข้อมูล
- “a”      ทำการเปิดเพิ่มข้อมูลเพื่อทำการเขียนข้อมูลใหม่ต่อท้ายข้อมูลที่มีอยู่เดิม
- “r+”     ทำการเปิดเพิ่มข้อมูลที่มีอยู่แล้วเพื่อทำการแก้ไขข้อมูล
- “w+”     ทำการสร้างเพิ่มข้อมูลใหม่และเปิดเพิ่มข้อมูลนั้นเพื่อทำการอ่านและเขียนข้อมูล
- “a+”     ทำการเปิดเพิ่มข้อมูลเพื่อทำการเขียนข้อมูลใหม่ต่อท้ายข้อมูลที่มีอยู่เดิม และทำการสร้างเพิ่มข้อมูลใหม่หากไม่พบเพิ่มข้อมูลที่จะระบุ

หากขั้นตอนของการเปิดแฟ้มข้อมูลโดยฟังก์ชัน `fopen()` ทำงานไม่สำเร็จ อาจเนื่องจากไม่พบแฟ้มข้อมูลนั้นในกรณีที่ต้องการเปิดแฟ้มข้อมูลเพื่ออ่าน หรือข้อมูลในแผ่นเต็ม หรือชื่อแฟ้มข้อมูลยาวเกินไป เป็นต้น ค่าที่ส่งกลับจากฟังก์ชัน `fopen()` จะมีค่าเท่ากับว่าง (Null)

### 1.2 ฟังก์ชัน `fclose()`

เมื่อทำการประมวลผลแฟ้มข้อมูลเสร็จเรียบร้อยแล้ว สิ่งที่จะต้องทำคือการปิดแฟ้มข้อมูลนั้นเพื่อคืนค่าพื้นที่ในหน่วยความจำให้สามารถนำไปใช้งานอื่นต่อไปได้ รูปแบบของฟังก์ชัน `fclose()` มีดังนี้

```
fclose(fileptr);
```

โดยที่ `fileptr` คือตัวแปรพอยน์เตอร์ชี้ไปยังแฟ้มข้อมูลที่ต้องการปิด ทุกครั้งที่มีการเปิดแฟ้มข้อมูลด้วยฟังก์ชัน `fopen()` ควรจะใช้ฟังก์ชัน `fclose()` ด้วยเสมอ

### 1.3 ฟังก์ชัน `fgetc()`

ฟังก์ชันนี้ทำงานคล้ายกับฟังก์ชัน `getchar()` ต่างกันที่ฟังก์ชัน `getchar()` จะทำการอ่านค่าอักขระทีละอักขระจากอุปกรณ์รับข้อมูลมาตรฐาน ส่วนฟังก์ชัน `fgetc()` จะทำการอ่านค่าอักขระทีละอักขระจากแฟ้มข้อมูล รูปแบบของฟังก์ชัน `fgetc()` เป็นดังนี้

```
ch = fgetc (fileptr) ;
```

โดยที่ `ch` เป็นตัวแปรที่รับค่าอักขระที่ได้จากการอ่านแฟ้มข้อมูล และ `fileptr` คือตัวแปรพอยน์เตอร์ชี้ไปยังแฟ้มข้อมูลที่ต้องการอ่านข้อมูล

### 1.4 ฟังก์ชัน `fputc()`

ฟังก์ชันนี้ใช้สำหรับการเขียนข้อมูลที่ละอักขระลงในแฟ้มข้อมูล เมื่อเขียนข้อมูลเสร็จ 1 อักขระ จะทำการเลื่อนตำแหน่งของพอยน์เตอร์ของแฟ้มข้อมูลไป 1 ตำแหน่ง รูปแบบของฟังก์ชัน `fputc()` เป็นดังนี้

```
fputc (ch, fileptr);
```

โดยที่ `ch` เป็นตัวแปรที่เก็บค่าอักขระที่ต้องการเขียนลงในแฟ้มข้อมูล และ `fileptr` คือตัวแปรพอยน์เตอร์ชี้ไปยังแฟ้มข้อมูลที่ต้องการเขียนข้อมูล

### 1.5 ฟังก์ชัน `fgets()`

ฟังก์ชันนี้ใช้สำหรับอ่านข้อมูลในรูปแบบของสตริงจากแฟ้มข้อมูล โดยในการอ่านค่าข้อมูลต้องทำการระบุขนาดของข้อมูลที่ต้องการอ่าน รูปแบบของฟังก์ชัน `fgets()` เป็นดังนี้

```
fgets (str, len, fileptr);
```

โดยที่ `str` เป็นตัวแปรที่ใช้สำหรับเก็บค่าสตริงที่ได้จากการอ่านข้อมูล `len` คือตัวแปรที่ใช้สำหรับกำหนดขนาดของข้อมูลที่ต้องการอ่าน และ `fileptr` คือตัวแปรพอยน์เตอร์ชี้ไปยังแฟ้มข้อมูลที่ต้องการอ่าน



### 1.6 ฟังก์ชัน fputs()

ฟังก์ชันนี้ใช้สำหรับการเขียนข้อมูลในรูปแบบของสตริงลงในแฟ้มข้อมูล รูปแบบของฟังก์ชัน fputs () เป็นดังนี้

```
fputs (str , fileptr);
```

โดยที่ str เป็นตัวแปรที่ใช้สำหรับเก็บค่าสตริงที่ต้องการบันทึก และ fileptr คือตัวแปรพอยน์เตอร์ชี้ไปยังแฟ้มข้อมูลที่ต้องการเขียน

### 1.7 ฟังก์ชัน fscanf() และ fprintf()

หากต้องการอ่านหรือเขียนข้อมูลโดยกำหนดรูปแบบของการประมวลผลนั้น สามารถทำได้โดยเรียกใช้ฟังก์ชัน fscanf () เพื่อการอ่านข้อมูล และเรียกใช้ฟังก์ชัน fprintf () เพื่อการเขียนข้อมูล รูปแบบของทั้งสองฟังก์ชันเป็นดังนี้

```
fscanf (fileptr , format , arg1,arg2, ...);
```

```
fprintf (fileptr , format , arg1,arg2, ...);
```

fileptr คือ ตัวแปรพอยน์เตอร์ชี้ไปยังแฟ้มข้อมูลที่ต้องการอ่านหรือเขียน

*format* คือ รูปแบบของการอ่านและเขียนข้อมูล

*arg1,arg2,...* คือ ตัวแปรที่ต้องการส่งค่าเพื่ออ่านหรือเขียนข้อมูล

### 1.8 ฟังก์ชัน feof()

การเก็บข้อมูลในแฟ้มข้อมูลทุกแฟ้มนั้นจะมีการเก็บอักขระพิเศษที่ไม่สามารถมองเห็นได้คือ EOF (End Of File) เพื่อเป็นลึกลับบอกให้ทราบว่า เป็นจุดสิ้นสุดของแฟ้มข้อมูล ฟังก์ชันที่ใช้ตรวจสอบว่าตำแหน่งของไฟล์พอยน์เตอร์ปัจจุบันเป็นตำแหน่งสิ้นสุดของแฟ้มข้อมูลหรือไม่คือฟังก์ชัน feof ()

รูปแบบการใช้งานฟังก์ชันนี้คือ

```
retvalue = feof (fileptr);
```

fileptr คือ ตัวแปรพอยน์เตอร์ชี้ไปยังแฟ้มข้อมูล

retvalue คือ ค่าที่คืนกลับมาเพื่อบอกให้ทราบว่าไฟล์พอยน์เตอร์อยู่ที่ตำแหน่ง EOF หรือไม่

ถ้า retvalue = 0 แสดงว่าไฟล์พอยน์เตอร์ไม่ได้ชี้อยู่ที่ตำแหน่ง EOF

ถ้า retvalue ไม่เท่ากับ 0 แสดงว่าไฟล์พอยน์เตอร์ชี้อยู่ที่ตำแหน่ง EOF

## 2. การบันทึกข้อมูลลงแฟ้มข้อมูล

ในการบันทึกข้อมูลลงแฟ้มข้อมูลนั้น สิ่งที่ผู้เขียนโปรแกรมจำเป็นต้องพิจารณาคือรูปแบบของการเปิดแฟ้มข้อมูล ซึ่งถ้าหากระบุรูปแบบของการเปิดแฟ้มที่ไม่ถูกต้องวิธี จะส่งผลทำให้การทำงานผิดพลาดประสงค์ เช่น หากต้องการสร้างแฟ้มข้อมูลใหม่ ให้ระบุรูปแบบการเปิดแฟ้มข้อมูลเป็นแบบ "w" และหากต้องการทำการเขียนข้อมูลใหม่ต่อท้ายข้อมูลที่มีอยู่เดิม ให้ระบุรูปแบบการเปิดแฟ้มข้อมูลเป็นแบบ "a" เป็นต้น แสดงตัวอย่างการทำงานดังตัวอย่างที่ 8.1

### ตัวอย่างที่ 8.1 การรับข้อมูลจากแผงแป้นอักขระ (keyboard) และบันทึกลงแฟ้มข้อมูล

```
#include <stdio.h>
/* Create a file form keyboard */
void main() {
 FILE *fileptr ; /* declared a pointer to an output file */
 char ch;

 if ((fileptr = fopen("data.dat", "w")) != NULL) { /* open file for writing */
 while ((ch = getchar ()) != "\n") /* get character from keyboard */
 fputc(ch, fileptr); /* write a character into file */
 fclose(fileptr); /* close file */
 } else
 printf (" Error in file open ");
}
```

การทำงานของตัวอย่างที่ 8.1 เริ่มจากขั้นตอนของการเปิดแฟ้มข้อมูล data.dat โปรแกรมจะทำการสร้างแฟ้มข้อมูลใหม่ เนื่องจากโปรแกรมระบุรูปแบบการเปิดแฟ้มข้อมูลเป็นแบบ "w" หากเกิดข้อผิดพลาดขึ้นในขั้นตอนนี้ ซึ่งอาจเกิดจากแผ่นดิสก์เต็ม หรือไม่มีแผ่นดิสก์ในช่องอ่านข้อมูลที่ระบุ เป็นต้น โปรแกรมจะแสดงข้อความว่า "Error in file open" บนจอภาพ

หากการเปิดสร้างแฟ้มข้อมูล หรือเปิดแฟ้มข้อมูลใหม่ทำได้สำเร็จ โปรแกรมจะเริ่มทำการรับข้อมูลที่ละอักขระจากแป้นพิมพ์ โดยกำหนดให้สร้างแฟ้มข้อมูลขึ้นมาใหม่ชื่อว่า data.dat และให้ตัวแปร ch เป็นตัวแปรที่ใช้สำหรับการรับอักขระที่ผู้ใช้ทำการพิมพ์ จากนั้นทำการบันทึกอักขระนั้นลงในแฟ้มข้อมูล โปรแกรมจะทำการรับข้อมูลจนกระทั่งผู้ใช้เคาะแป้น Enter การทำงานของโปรแกรมจะหยุดลง

### 3. การอ่านข้อมูลจากแฟ้มข้อมูล

ในการอ่านข้อมูลลงในแฟ้มข้อมูลนั้น สิ่งที่คุณเขียนโปรแกรมต้องพิจารณาเช่นเดียวกับการบันทึกข้อมูลลงในแฟ้มข้อมูลคือรูปแบบของการเปิดแฟ้มข้อมูล ถ้าหากการประมวลผลข้อมูลนั้นต้องการอ่านข้อมูลขึ้นมาแสดงผลเพียงอย่างเดียว รูปแบบการเปิดแฟ้มข้อมูลที่ระบุควรเป็นแบบ "r" และหากต้องการทำการอ่านข้อมูลและทำการแก้ไขข้อมูลในแฟ้มนั้นด้วย ก็ควรระบุรูปแบบการเปิดแฟ้มข้อมูลเป็นแบบ "r+" เป็นต้น แสดงการใช้งานแฟ้มข้อมูลดังตัวอย่างที่ 8.2 ถึง 8.4

## ตัวอย่างที่ 8.2 การอ่านข้อมูลจากแฟ้มข้อมูล

```
#include <stdio.h>
/* Program for reading data from file */
void main() {
 FILE *fileptr ; /* declared a pointer to an input file */
 int ch;
 if ((fileptr = fopen("data.dat", "r")) != NULL) {
 while ((ch = fgetc (fileptr)) != EOF)
 putchar(ch); /* display a character on screen */
 fclose(fileptr);
 } else
 printf (" Couldn't open file data.dat ");
}
```

การทำงานของตัวอย่างที่ 8.2 เริ่มต้นจากขั้นตอนของการเปิดแฟ้มข้อมูล data.dat เช่นเดียวกับตัวอย่างที่ 8.1 ต่างกันตรงที่โปรแกรมนี้มีวัตถุประสงค์การทำงานเพื่อการอ่านข้อมูล ดังนั้นรูปแบบการเปิดแฟ้มข้อมูลที่ระบุจึงเป็นแบบ "r" เนื่องจากต้องการอ่านข้อมูลเพียงอย่างเดียว หากเกิดข้อผิดพลาดขึ้นในขั้นตอนไม่มีแฟ้มข้อมูลชื่อว่า data.dat อยู่ในสารบบ (directory) ปัจจุบันที่โปรแกรมทำงาน หรือ ไม่มีแผ่นดิสก์ในช่องอ่านข้อมูลที่ระบุ เป็นต้น โปรแกรมจะแสดงข้อความว่า "couldn't open file data.dat" บนจอภาพ

หากการเปิดแฟ้มข้อมูลทำได้สำเร็จ โปรแกรมจะทำการอ่านข้อมูลจากแฟ้มข้อมูล data.dat ทีละอักขระ และนำมาแสดงผลทางจอภาพ สิ่งที่น่าสนใจสำหรับโปรแกรมนี้คือการกรรมวิธีการอ่านข้อมูลจากแฟ้มข้อมูล การทำงานจะวนอ่านข้อมูลที่ละอักขระจนกระทั่งถึงจุดสิ้นสุดของแฟ้มข้อมูล ซึ่งทำการตรวจสอบจุดสิ้นสุดโดยคำสั่ง

```
while ((ch = fgetc (fileptr)) != EOF)
```

จะเห็นว่าวิธีการตรวจสอบจุดสิ้นสุดของแฟ้มข้อมูลสามารถทำได้โดยตรวจสอบว่าอักขระตัวปัจจุบันที่อ่านเข้ามาเก็บไว้ในตัวแปร ch มีค่าเท่ากับ EOF (ซึ่งมีความหมายตรงกับคำว่า End of file) หรือไม่

## ตัวอย่างที่ 8.3 การคัดลอกแฟ้มข้อมูล

```
#include <stdio.h>
/* Program for reading each character from one file and write it to another */
void main() {
 FILE *inp_fp, out_fp ; /* declared a pointer to an input and output file */
 int ch;
 if ((inp_fp = fopen("data.dat", "r")) != NULL) {
 if ((out_fp = fopen("newdata.dat", "w")) != NULL) {
 while ((ch = fgetc (inp_fp)) != EOF)
 fputc(ch, out_fp); /* copy a character to new file */
 }
 }
```

```

 fclose(inp_fp);
 fclose(out_fp);
 } else
 printf(" Couldn't create file newdata.dat ");
} else
 printf(" Couldn't open file data.dat ");
}

```

---

การทำงานของโปรแกรมตัวอย่างที่ 8.3 จะเปิดแฟ้มข้อมูลต้นฉบับที่จะทำการอ่านขึ้นมาคือแฟ้ม data.dat โดยอ่านข้อมูลขึ้นมาทีละหนึ่งตัวอักษรด้วยฟังก์ชัน fgetc ( ) และมีการเปิดแฟ้มข้อมูลใหม่ขึ้นมาคือแฟ้ม newdata.dat เพื่อทำการสำเนาข้อมูลจากแฟ้ม data.dat ทีละหนึ่งตัวอักษรด้วยฟังก์ชัน fputc ( ) การสำเนาข้อมูลนี้จะทำสำเนาทีละหนึ่งตัวอักษรจนกว่าจะพบอักขระ EOF จากการอ่านแฟ้ม data.dat

---

#### ตัวอย่างที่ 8.4 แสดงการใช้งานฟังก์ชัน fscanf ( ) และ fprintf ( )

---

```

/* Copy a series of integers from a source file to a destination file. The file name are given as
 command line arguments. */
#include <stdio.h>
#define READONLY "r" /* declared constant for reading status */
#define WRITEONLY "w" /* declared constant for writing status */
void main (int argc, char *argv[]) {
 FILE *fpin, *fpout;
 int data, status;
 if (argc != 3)
 printf ("Usage : %s file1 file2\n", argv[0]);
 else if ((fpin = fopen (argv[1], READONLY)) == NULL)
 printf ("%s : cannot open %s\n", argv[0], argv[1]);
 else if ((fout = fopen (argv[2], WRITEONLY)) == NULL) {
 printf ("%s : cannot open %s\n", argv[0], argv[2]);
 fclose (fpin);
 } else {
 while ((status = fscanf (fpin, "%d", &data)) != EOF) {
 if (status != 1) {
 printf ("Failure on file read\n");
 break;
 }
 if (fprintf (fpout, "%d\n", data) < 0) {
 printf ("Failure on file write\n");
 break;
 }
 }
 }
}

```

```

 }
}
fclose (fpin);
fclose (fpout);
}

```

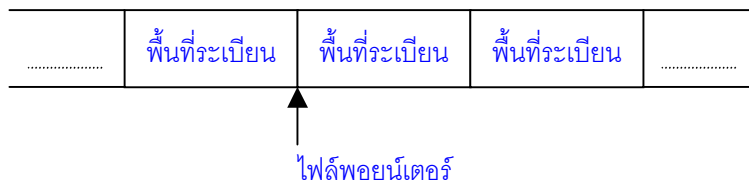
การทำงานของโปรแกรมนี้จะทำการสำเนาข้อมูลที่เก็บข้อมูลจำนวนเต็มจากแฟ้มหนึ่งไปยังอีกแฟ้มหนึ่ง โดยให้ผู้ใช้กำหนดชื่อแฟ้มข้อมูลต้นฉบับและแฟ้มข้อมูลปลายทางในลักษณะของคอมมานไลน์อาร์กิวเมนต์ โดยจะมีการตรวจสอบจำนวนของอาร์กิวเมนต์ ที่ส่งเข้าจากการเรียกใช้โปรแกรม หากส่งจำนวนอาร์กิวเมนต์ เข้ามาถูกต้องครบ 3 ตัว คือ ชื่อโปรแกรม แฟ้มต้นฉบับ แฟ้มปลายทาง จะมีการเปิดแฟ้มต้นฉบับ ซึ่งหากเปิดแฟ้มได้ก็จะทำการเปิดแฟ้มปลายทางที่จะทำสำเนา ในกรณีนี้จะเห็นว่าหากไม่สามารถเปิดแฟ้มปลายทางได้ จะมีการแจ้งข้อความแสดงความผิดพลาดและมีการสั่งปิดแฟ้มต้นฉบับ เพราะก่อนหน้านี้มีการเปิดแฟ้มต้นฉบับเพราะฉะนั้นเมื่อจะจบการทำงานจึงต้องมีการปิดแฟ้มต้นฉบับก่อน แต่หากสามารถเปิดแฟ้มปลายทางได้เรียบร้อย ก็ทำการอ่านแฟ้มต้นฉบับด้วยฟังก์ชัน `fscanf ( )` หากข้อมูลไม่ถูกต้องฟังก์ชันจะคืนค่า 1 ให้กับ `status` เพื่อบอกให้รู้ว่าไม่สามารถอ่านข้อมูลได้ แต่หากอ่านข้อมูลได้ก็จะนำข้อมูลนั้นไปเขียนลงในแฟ้มปลายทางด้วยฟังก์ชัน `fprintf ( )` จนกว่าจะอ่านข้อมูลถึงตำแหน่ง EOF จึงจะหยุดทำงาน

ในทีนี้สิ่งที่ต้องระวังคือก่อนที่จะอ่านแฟ้มข้อมูลใด ๆ ขึ้นมาใช้งานได้ จะต้องทราบว่าแฟ้มข้อมูลนั้นจัดเก็บข้อมูลในลักษณะอย่างไร หากไม่ทราบรูปแบบการจัดเก็บข้อมูลก็จะไม่สามารถอ่านแฟ้มข้อมูลนั้นมาใช้งานได้

#### 4. แฟ้มข้อมูลแบบเข้าถึงโดยตรง (Direct Access File)

การใช้งานแฟ้มข้อมูลจากตัวอย่างที่ผ่านมาข้างต้น จะเห็นว่าจะต้องอ่านข้อมูลขึ้นมาทีละลำดับจนกว่าจะถึงตำแหน่งสุดท้ายของแฟ้มข้อมูล (EOF) ซึ่งเป็นการเข้าถึงข้อมูลแบบลำดับ ข้อเสียของแฟ้มประเภทนี้คือจะต้องอ่านข้อมูลตั้งแต่ตำแหน่งแรกจนกว่าจะถึงตำแหน่งที่ต้องการเสมอ ซึ่งทำให้เสียเวลา

แฟ้มข้อมูลอีกประเภทหนึ่งที่อนุญาตให้มีการเข้าถึงข้อมูลตำแหน่งที่ต้องการได้คือ แฟ้มข้อมูลแบบเข้าถึงโดยตรง การใช้แฟ้มประเภทนี้จะต้องรู้โครงสร้างของข้อมูลที่เก็บอยู่ในแฟ้มนั้น ซึ่งปกติจะเก็บข้อมูลเป็นระเบียบต่อเนื่องกันไป โดยที่แต่ละระเบียนมีโครงสร้างข้อมูลเหมือนกันเสมอ ด้วยวิธีนี้จะได้ว่าแต่ละระเบียนมีขนาดของข้อมูลที่เท่ากัน เพราะฉะนั้นการอ่านหรือเขียนข้อมูลที่ตำแหน่งที่ต้องการจะสามารถคำนวณตำแหน่งที่แน่นอนได้เสมอ โดยการเลื่อนไฟล์พอยน์เตอร์ไปยังตำแหน่งเริ่มต้นของระเบียนที่ต้องการอ่านหรือเขียนข้อมูลนั้น แสดงดังรูปที่ 8.1



รูปที่ 8.1 แสดงไฟล์พอยน์เตอร์ชี้ไปยังตำแหน่งเริ่มต้นของระเบียนในแฟ้มข้อมูลแบบเข้าถึงโดยตรง

แฟ้มข้อมูลที่มีการเข้าถึงโดยตรงหากลองเปิดแฟ้มข้อมูลดูเนื้อหาภายใน จะพบว่าไม่สามารถอ่านข้อมูลดังกล่าวได้ แฟ้มในลักษณะนี้จะใช้ฟังก์ชัน `fread ( )` หรือ `fwrite ( )` ในการอ่านและบันทึกข้อมูล ซึ่งจะกล่าวถึงต่อไป แฟ้มในลักษณะนี้จะ

เรียกว่า ไบนารีไฟล์ (Binary File) จะแตกต่างจากแฟ้มในตัวอย่าง 8.1 หรือ 8.2 ซึ่งสามารถเปิดแฟ้มเพื่อดูเนื้อหาภายในได้ แฟ้มที่เก็บข้อมูลในลักษณะที่คนทั่วไปสามารถอ่านได้นี้เรียกว่า เทกซ์ไฟล์ (Text File) คำสั่งเพิ่มเติมสำหรับการจัดการแฟ้มข้อมูลแบบเข้าถึงโดยตรงได้แก่

#### 4.1 ฟังก์ชัน ftell( )

หากต้องการทราบตำแหน่งของพอยน์เตอร์ที่ชี้ตำแหน่งในแฟ้มข้อมูลปัจจุบัน สามารถทำได้โดยเรียกใช้ฟังก์ชัน ftell( ) โดยมีรูปแบบดังนี้

```
pos = ftell(fileptr);
```

fileptr คือ ตัวแปรพอยน์เตอร์ที่ชี้ไปยังแฟ้มข้อมูล และ pos คือ ตัวแปรสำหรับรับค่าตำแหน่งของไฟล์พอยน์เตอร์ โดยตัวแปรนี้ต้องถูกกำหนดให้เป็นตัวแปรแบบ long int

#### 4.2 ฟังก์ชัน fseek( )

เมื่อต้องการเข้าถึงข้อมูลโดยตรง ณ ตำแหน่งใดตำแหน่งหนึ่งโดยเจาะจงในทันที สามารถทำได้โดยเรียกใช้งานฟังก์ชัน fseek ซึ่งมีรูปแบบดังนี้

```
fseek(fileptr, offset , refvalue);
```

fileptr คือ ตัวแปรพอยน์เตอร์ที่ชี้ไปยังแฟ้มข้อมูล

offset คือ ตัวแปรเก็บค่าตำแหน่งที่ต้องการให้พอยน์เตอร์ชี้ไปนับห่างจากตำแหน่งที่ระบุ

โดยตัวแปรนี้เก็บค่าข้อมูลในรูปแบบ long integer

refvalue คือ ตัวแปรเก็บการระบุตำแหน่งของการค้นหาข้อมูล

ถ้ากำหนดค่าให้ refvalue = 1 หรือ SEEK\_SET จะทำการนับตำแหน่งจากต้นแฟ้มข้อมูล

ถ้า refvalue = 2 หรือ SEEK\_CUR จะทำการนับตำแหน่งจากตำแหน่งปัจจุบัน

ถ้า refvalue = 3 หรือ SEEK\_END จะทำการนับตำแหน่งจากท้ายแฟ้มข้อมูล

#### 4.3 ฟังก์ชัน rewind ( )

ฟังก์ชันนี้ใช้สำหรับการเลื่อนไฟล์พอยน์เตอร์กลับไปตำแหน่งเริ่มต้นของแฟ้มข้อมูล ซึ่งคำสั่งนี้จะมีผลของการทำงาน เหมือนกับการใช้คำสั่ง fseek ( fp, 0L, SEEK\_SET ) การเรียกใช้งานฟังก์ชัน rewind สามารถทำได้ดังนี้

```
rewind (fileptr)
```

fileptr คือ ตัวแปรพอยน์เตอร์ที่ชี้ไปยังแฟ้มข้อมูล

### 5. แฟ้มข้อมูลแบบเรคคอร์ด

ในการนำข้อมูลไปใช้งานในการทำงานจริง ข้อมูลส่วนใหญ่มักเป็นข้อมูลแบบเรคคอร์ด เช่น การจัดเก็บแฟ้มข้อมูลสำหรับจัดเก็บข้อมูลนักศึกษา สิ่งที่เราสนใจจัดเก็บอาจได้แก่ รหัสนักศึกษา ชื่อนักศึกษา ฯลฯ แต่ละรายการที่กล่าวมาจัดเป็นฟิลด์หนึ่งในเรคคอร์ด หากเราต้องการจัดเก็บข้อมูลเหล่านี้ อาจทำได้ดังตัวอย่างที่ 8.5 แต่ก่อนอื่นเราควรรมาทำความเข้าใจกับฟังก์ชันสำคัญอีก 2 ฟังก์ชันที่ใช้สำหรับการจัดการแฟ้มข้อมูลแบบเรคคอร์ด คือ

### 5.1 ฟังก์ชัน fwrite ( ) และ fread ( )

หากต้องการอ่านหรือเขียนข้อมูล

โดยกำหนดรูปแบบของการประมวลผลนั้น สามารถทำได้โดยเรียกใช้ฟังก์ชัน fread ( ) เพื่อการอ่านข้อมูล และเรียกใช้ฟังก์ชัน fwrite ( ) เพื่อการเขียนข้อมูล การใช้งานฟังก์ชันทั้ง 2 จะคล้ายกับการใช้งานฟังก์ชัน fscanf( ) และ fprintf( ) รูปแบบของทั้งสองฟังก์ชันเป็นดังนี้

```
fwrite(dataptr , sizeof(record struct), 1, fileptr);
```

```
fread(dataptr , sizeof(record struct), 1, fileptr);
```

*dataptr* คือ ตัวแปรพอยน์เตอร์ชี้ไปยังที่เก็บข้อมูล

*record struct* คือ ชื่อโครงสร้างข้อมูลเรคคอร์ดที่ต้องการอ่านหรือเขียน

fileptr คือ ตัวแปรพอยน์เตอร์ชี้ไปยังแฟ้มข้อมูลที่ต้องการอ่านหรือเขียน

#### ตัวอย่างที่ 8.5 แสดงการใช้แฟ้มข้อมูลแบบเรคคอร์ด

---

```
#include <stdio.h>

#define OK 1
#define ERR 0
#define ARR_SIZE 3

typedef struct {
 char ID[8];
 char Name[21];
} Student;

int writeData(Student pStudent[]) {
 FILE *fptr;
 int i;
 if ((fptr = fopen("student.dat", "w")) != NULL) {
 for (i = 0; i < ARR_SIZE; i++)
 fwrite(&(pStudent[i]), sizeof(Student), 1, fptr);
 fclose(fptr);
 return (OK);
 } else
 return (ERR);
}

int readData(Student *tStudent, int position) {
 FILE *fptr;
 int size;
```

```

if ((fptr = fopen("student.dat", "r")) != NULL) {
 fseek(fptr, 0L, SEEK_END); /* go to end of file */
 size = ftell(fptr);
 if ((position > 0) && (size/sizeof(Student) >= position)) {
 fseek(fptr, 0L, SEEK_SET); /* go to begin of file */
 fseek(fptr, (position-1)*sizeof(Student), SEEK_SET);
 fread(tStudent, sizeof(Student), 1, fptr);
 printf("\nID %s, Name %s", tStudent->ID, tStudent->Name);
 fclose(fptr);
 return (OK);
 } else {
 printf("\nYour seek position is out of range");
 return (ERR);
 }
} else
 return (ERR);
}

void main () {
 Student aStudent[ARR_SIZE];
 Student fStudent;
 int i, code, position;

 for (i = 0; i < ARR_SIZE; i++) {
 printf("\nStudent Data : ");
 scanf("%s %s", &(aStudent[i].ID), &(aStudent[i].Name));
 }

 if (code = writeData(aStudent)) {
 printf("\nSelect position : ");
 scanf("%d", &position);
 if (code = readData(&fStudent, position)) {
 printf("\nID : %s, Name : %s", fStudent.ID, fStudent.Name);
 }
 }
}

```

---

ตัวอย่างที่ 8.5 นี้จะทำการอ่านข้อมูลรหัสและชื่อของนักศึกษาเข้ามาทีละคนจากทางแป้นพิมพ์ แล้วทำการเก็บไว้ในตัวแปรอาเรย์ ชื่อ aStudent จากนั้นก็จะทำการเขียนข้อมูลลงไปในพื้นที่ข้อมูลชื่อ student.dat โดยเรียกใช้งานฟังก์ชัน writedata ซึ่งเป็นฟังก์ชันที่เขียนขึ้นมาเอง

เมื่อทำการเขียนข้อมูลเสร็จโปรแกรมจะทำการตรวจสอบข้อผิดพลาดในการเขียนข้อมูล จากประโยค



```
if (code = writeData(aStudent))
```

ถ้าหากในขั้นตอนของการเขียนข้อมูลไม่เกิดข้อผิดพลาด โปรแกรมจะทำงานต่อไปโดยจะรอรับข้อมูลตำแหน่งของเรคคอร์ดที่ผู้ใช้ต้องการให้แสดง โดยค่าที่ป้อนนี้สามารถป้อนได้ตั้งแต่ 0-2 (เนื่องจากเรากำหนดขนาดของการรับข้อมูลในอาเรย์ไว้ 3 คน) เมื่อผู้ใช้ทำการป้อนตำแหน่งเรคคอร์ดที่ต้องการเรียบร้อยแล้ว โปรแกรมแสดงค่าข้อมูล ณ ตำแหน่งที่ผู้ใช้ระบุ โดยทำการเรียกใช้งานฟังก์ชัน readdata ซึ่งเป็นฟังก์ชันที่เขียนขึ้นมาเองเช่นกัน ตัวอย่างการทำงานเช่น หากผู้ใช้ป้อนข้อมูลที่ตำแหน่งนี้เป็น 0 โปรแกรมจะทำการแสดงข้อมูลเรคคอร์ดแรกที่บันทึกไว้ทางจอภาพ เป็นต้น

การค้นหาข้อมูลในตัวอย่างนี้จะเป็นการค้นหาโดยใช้การเข้าถึงโดยตรง (Direct Access) โดยใช้งานฟังก์ชัน fseek ที่ได้อธิบายวิธีการใช้งานในหัวข้อก่อนหน้านี้แล้ว

## 6. อุปกรณ์มาตรฐานในการนำเข้าและแสดงผลข้อมูล

ในขณะที่โปรแกรมภาษาซีเริ่มต้นทำงาน จะมีแฟ้มข้อมูล 3 แฟ้มเปิดขึ้นโดยอัตโนมัติโดยระบบ แฟ้มเหล่านี้คือ แฟ้มที่ทำหน้าที่นำเข้าข้อมูลมาตรฐาน (Standard Input) แฟ้มที่ทำหน้าที่แสดงข้อมูลมาตรฐาน (Standard Output) แฟ้มที่แสดงข้อความการผิดพลาดของการทำงานมาตรฐาน (Standard Error) โดยมีตัวชี้แฟ้มข้อมูลรองรับการทำงานของแฟ้มทั้ง 3 ซึ่งกำหนดไว้ในแฟ้ม stdio.h

แฟ้มเหล่านี้จะติดต่อกับอุปกรณ์มาตรฐานต่าง ๆ โดยทั่วไปคือ แฟ้มที่ทำหน้าที่นำเข้าข้อมูลมาตรฐานจะเป็นการรับข้อมูลจากแป้นพิมพ์ ส่วนแฟ้มที่ทำหน้าที่แสดงข้อมูลมาตรฐานและแฟ้มที่แสดงข้อความการผิดพลาดของการทำงานมาตรฐานจะเป็นการแสดงผลทางจอภาพ การเรียกใช้ฟังก์ชันที่เกี่ยวข้องกับอุปกรณ์ดังกล่าวสามารถไม่เขียนระบุแฟ้มมาตรฐานเหล่านี้ก็ได้ โดยการเรียกใช้ฟังก์ชันที่ทำงานกับอุปกรณ์มาตรฐาน หรือหากต้องการระบุแฟ้มก็สามารถเรียกใช้ผ่านฟังก์ชันที่เกี่ยวข้องกับฟังก์ชันที่ทำงานกับแฟ้มข้อมูล การทำงานกับแฟ้มที่ทำหน้าที่นำเข้าข้อมูลมาตรฐานสามารถระบุแทนด้วย stdin เช่น

```
fscanf (stdin, "%d", &number);
```

จะมีผลการทำงานเท่ากับคำสั่งของฟังก์ชันมาตรฐาน

```
scanf ("%d", &number);
```

การใช้คำสั่งที่ทำงานกับแฟ้มที่ทำหน้าที่แสดงข้อมูลมาตรฐานสามารถระบุแทนด้วย stdout

```
fprintf (stdout, "My first program\n");
```

จะมีผลการทำงานเท่ากับคำสั่งของฟังก์ชันมาตรฐาน

```
printf ("My first program");
```

ส่วนการใช้คำสั่งที่ทำงานกับแฟ้มที่ทำหน้าที่แสดงข้อความการผิดพลาดของการทำงานมาตรฐานสามารถระบุแทนด้วย stderr เช่น การบันทึกความผิดพลาดของการทำงานดังนี้

```
if ((fp = fopen (argv[1], READONLY)) == NULL)
```

```
fprintf (stderr, "%s : cannot open %s\n", argv[0], argv[1]);
```

## 7. การรองรับความผิดพลาดจากการทำงาน (Error Handling)

การทำงานของโปรแกรมภาษาซีปกติจะเริ่มต้นทำงานที่ฟังก์ชัน main ( ) ตั้งแต่เริ่มต้น จนกระทั่งถึงคำสั่งสุดท้าย แต่หากต้องการจบโปรแกรมที่จุดใดจุดหนึ่งก่อนจบฟังก์ชัน main ( ) เช่น จบโปรแกรมเมื่อเกิดข้อผิดพลาดขึ้น จะสามารถทำได้ด้วยคำสั่ง

```
exit (n);
```

โดยที่ n คือ เลขจำนวนเต็มที่แสดงความผิดพลาด ซึ่งสามารถเข้าใจได้โดยโปรแกรมเมื่อผู้เขียนโปรแกรมนั้นเป็นผู้ระบุ โดยทั่วไปการทำงานใด ๆ ในภาษาซีหากการทำงานสำเร็จตามขั้นตอนมักจะแทน n ด้วยค่า 0 แต่หากเป็นความผิดพลาดอื่นอาจแทนด้วยค่าใด ๆ ที่ไม่ใช่ 0 ซึ่งผู้เขียนโปรแกรมสามารถกำหนดเพื่อความเข้าใจของตนเองได้ เช่น ให้มีค่าเป็น -1 หากส่งอาร์กิวเมนต์ ในการใช้โปรแกรมไม่ครบ หรือให้คืนค่า -2 หากไม่สามารถเปิดแฟ้มข้อมูลที่ระบุได้ หรืออาจจะใช้ EXIT\_FAILURE และ EXIT\_SUCCESS ที่มีกำหนดในแฟ้ม stdlib.h ก็ได้ แสดงดังตัวอย่างที่ 8.6

### ตัวอย่างที่ 8.6 แสดงการใช้คำสั่ง exit ( )

---

```
#include <stdio.h>
#include <stdlib.h>
typedef int Character; /* character input */
#define READONLY "r"
#define WRITEONLY "w"
void syserr (int, char *, char *);
char *programname;
int main (int argc, char *argv[]) {
 FILE *fpsource, *fpdestination;
 Character c;
 programname = argv[0];
 if (argc != 3)
 syserr (1, "Usage : %s file1 file2\n", programname);
 else if ((fpsource = fopen (argv[1], READONLY)) == NULL)
 syserr (2, "Cannot open %s\n", argv[1]);
 else if ((fpdestination = fopen (argv[2], WRITEONLY)) == NULL)
 syserr (3, "Cannot open %s\n", argv[2]);
 else {
 while ((c = fgetc (fpsource)) != EOF) /* loop until done */
 if (fputc (c, fpdestination) == EOF)
 syserr (4, "Error in writing to %s\n", argv[2]);
 if (!feof (fpsource)) /* all done? */
 syserr (5, "Error in reading %s\n", argv[1]);
 exit (EXIT_SUCCESS);
 }
}
```

```

 }
}
void syserr (int errcode, char *message, char *argument) {
 fprintf (stderr, "%s [%2d] : ", programname, errcode);
 fprintf (stderr, message, argument);
 exit (EXIT_FAILURE);
}

```

การทำงานของโปรแกรมตัวอย่างจะมีฟังก์ชันเพื่อแสดงความผิดพลาดที่เกิดขึ้นคือ `syserr ( )` เป็นฟังก์ชันที่รองรับการทำงานผิดพลาดใด ๆ ที่เกิดขึ้น การรองรับความผิดพลาดของโปรแกรมจะช่วยให้โปรแกรมเมอร์สามารถตรวจหาความผิดพลาดจากการทำงานของโปรแกรมได้ง่ายยิ่งขึ้น ซึ่งเป็นสิ่งที่สมควรจะทำ จากโปรแกรมตัวอย่างเป็นการอ่านข้อมูลจากแฟ้มต้นฉบับและเขียนลงยังแฟ้มข้อมูลปลายทาง มีการตรวจความผิดพลาดทั้งหมด 5 จุดคือ การส่งอาร์กิวเมนต์ ในการเรียกใช้โปรแกรมไม่ครบ การที่ไม่สามารถเปิดแฟ้มต้นฉบับ การที่ไม่สามารถเปิดแฟ้มปลายทาง การที่มีความผิดพลาดในการเขียนข้อมูลลงยังแฟ้มปลายทาง และการที่มีความผิดพลาดในการอ่านแฟ้มต้นฉบับ

## 8. ตัวอย่างเพิ่มเติม

ตัวอย่างเพิ่มเติมเกี่ยวกับการใช้แฟ้มข้อมูล แสดงดังตัวอย่างที่ 8.7 ถึง 8.12

**ตัวอย่างที่ 8.7** หากกำหนดให้ข้อมูลในแฟ้มข้อมูล `a1.data` เป็นข้อมูลตัวเลขจำนวนเต็มดังตัวอย่าง

```

1
2
3
4
5

```

ให้เขียนโปรแกรมเพื่อหาผลรวมของข้อมูลจากแฟ้มข้อมูล `a1.data`

```

#include <stdio.h>
int main() {
 FILE *fileptr;
 int data, sum=0;
 if ((fileptr = fopen("a1.data", "r"))==NULL) {
 printf("\nCan not open a1.dat");
 exit(-1);
 }
 fscanf(fileptr, "%d", &data);
 while (!feof(fileptr)) {
 sum += data;
 fscanf(fileptr, "%d", &data);
 }
}

```

```

fclose(fileptr);
printf("\nSum = [%d]\n\n", sum);
return(0);
}

```

**ตัวอย่างที่ 8.8** หากกำหนดให้ข้อมูลในแฟ้มข้อมูล a2.data เป็นข้อมูลตัวเลขจำนวนเต็มดังตัวอย่าง

```

1 2 3 4 5
6 7 8 9 10
11 12 13 14 15

```

ให้เขียนโปรแกรมเพื่อหาผลรวมของข้อมูลจากแฟ้มข้อมูล a2.data

```

#include <stdio.h>
int main() {
 FILE *fptr;
 int data, sum=0;
 if ((fptr=fopen("a2.data", "r"))==NULL) {
 printf("\nCan not open a2.data");
 exit(-1);
 }
 fscanf(fptr, "%d", &data);
 while (!feof(fptr)) {
 sum += data;
 fscanf(fptr, "%d", &data);
 }
 fclose(fptr);
 printf("\nSum [%d]", sum);
 return(0);
}

```

**ตัวอย่างที่ 8.9** หากกำหนดให้ข้อมูลในแฟ้มข้อมูล a2.data เป็นข้อมูลตัวเลขจำนวนเต็มดังตัวอย่าง

```

1 2 3 4 5
6 7 8 9 10
11 12 13 14 15

```

ให้เขียนโปรแกรมเพื่ออ่านข้อมูลจากแฟ้มข้อมูล a2.data มาเก็บในอาเรย์ 2 มิติ และให้เขียนข้อมูลเดิมต่อท้ายไฟล์ a2.data

```

#include <stdio.h>

#define ROW 3
#define COL 5

```

```

void readData(int[][COL]);
void printData(int[][COL]);
void writeData(int[][COL]);

int main() {
 int data[ROW][COL];
 readData(data);
 printData(data);
 writeData(data);
 return(0);
}

void writeData(int dat[][COL]) {
 FILE *optr;
 int i, j;
 if ((optr=fopen("a2.data", "a+"))==NULL) {
 printf("\nCan not open a2.data for write");
 exit(-2);
 }
 fprintf(optr, "\nNew Data");
 for (i=0; i<ROW; i++) {
 fprintf(optr, "\n");
 for (j=0; j<COL; j++) {
 fprintf(optr, "%d ", dat[i][j]);
 }
 }
 fclose(optr);
}

void readData(int dat[][COL]) {
 FILE *fptr;
 int i,j;
 if ((fptr=fopen("a2.data", "r"))==NULL) {
 printf("\nCan not open a2.data for read");
 exit(-1);
 }
 for (i=0; i<ROW; i++)
 for (j=0; j<COL; j++) {
 fscanf(fptr, "%d", &dat[i][j]);
 }
 fclose(fptr);
}

void printData(int data[][COL]) {

```

```

int i, j;
for (i=0; i<ROW; i++) {
 for (j=0; j<COL; j++) {
 printf("%5d%3s", data[i][j], "");
 }
 printf("\n");
}
}

```

---

**ตัวอย่างที่ 8.10** โปรแกรมเพื่ออ่านข้อมูลจากแฟ้ม a3.data ซึ่งเก็บข้อมูลชื่อและอายุ ดังตัวอย่าง

```

Kid 10
Somchai 15
Saijai 14
Ploy 12

```

เขียนโปรแกรมเพื่ออ่านข้อมูลและแสดงผลข้อมูลดังกล่าว และให้บันทึกข้อมูลที่ได้ออกทำไฟล์เดิม

---

```

#include <stdio.h>
#define MAX 10
typedef struct {
 char name[15];
 int age;
} Friend;
void readFriend(Friend pFriend[]) {
 FILE *fptr;
 char name[15];
 int age, count=0;

 if ((fptr=fopen("a3.data","r"))==NULL) {
 printf("\nCan not open a3.data for read");
 exit(-1);
 }
 fscanf(fptr,"%s %d", name, &age);
 while (!feof(fptr) && count < MAX) {
 strcpy(pFriend[count].name, name);
 pFriend[count].age = age;
 fscanf(fptr, "%s %d", name, &age);
 count++;
 }
 fclose(fptr);
}
/* แสดงข้อมูลในอาเรย์ทั้งหมดบนจอภาพ */

```

```

void printFriend1(Friend pFriend[]) {
 int i;
 printf("\n\nPrint Friend 1");
 for (i=0; i<MAX; i++) {
 printf("\nName [%s], age [%d]", pFriend[i].name, pFriend[i].age);
 }
}
/* แสดงเฉพาะสมาชิกของอาเรย์ที่มีข้อมูลอยู่บนจอภาพ */
void printFriend2(Friend pFriend[]) {
 int i=0;
 printf("\n\nPrint Friend 2");
 while (i<MAX && pFriend[i].name[0] != '\0') {
 printf("\nName [%s], age [%d]", pFriend[i].name, pFriend[i].age);
 i++;
 }
}
void writeFriend(Friend pFriend[]) {
 FILE *optr;
 int i=0;
 if ((optr=fopen("a3.data", "a+"))==NULL) {
 printf("\nCan not open a3.data for write");
 exit(-2);
 }
 fprintf(optr, "\nMy new data");
 while (i<MAX && pFriend[i].name[0] != '\0') {
 fprintf(optr, "\n%-20s %4d", pFriend[i].name, pFriend[i].age);
 i++;
 }
 fclose(optr);
}
int main() {
 Friend friend[MAX];
 memset(friend, '\0', sizeof(Friend)*MAX); /* กำหนดค่าเริ่มต้นให้กับข้อมูลในอาเรย์ทั้งหมด */
 readFriend(friend);
 printFriend1(friend);
 printFriend2(friend);
 printf("\n\n");
 writeFriend(friend);
 return(0);
}

```

---

**ตัวอย่างที่ 8.11** กำหนดให้อ่านข้อมูลรหัสนักศึกษาและคะแนนสอบของนักศึกษาโดยนำมาเก็บในตัวแปรโครงสร้างและให้บันทึกข้อมูลดังกล่าวลงในแฟ้มข้อมูล a4.data โดยใช้คำสั่ง fwrite( )

---

```
#include <stdio.h>

typedef struct {
 char id[8];
 float score;
} Score;

int main() {
 Score std;
 int repeat;
 FILE *fptr; /* Output file */
 if ((fptr=fopen("a4.data", "w"))==NULL) { /* Open file */
 printf("\nCan not open a4.data for write");
 exit(-1);
 }
 do {
 //Read data from user
 printf("\nEnter id : ");
 scanf("%s", std.id);
 printf("\nEnter score : ");
 scanf("%f", &std.score);
 if (fwrite(&std, sizeof(Score), 1, fptr) != 1) { /* Write data */
 printf("\nError writing data");
 exit(-3);
 }
 printf("\nRepeat (1-Yes/0-No)? "); /* Confirm repeat ? */
 scanf("%d", &repeat);
 } while (repeat);
 fclose(fptr); /* Close file */
 return(0);
}
```

---

**ตัวอย่างที่ 8.12** ให้อ่านข้อมูลจากไฟล์ a4.data และแสดงผลบนจอภาพ

---

```
#include <stdio.h>

typedef struct {
 char id[8];
 float score;
} Score;
```



```
int main() {
 Score std;
 int repeat;
 FILE *fptr; /* Output file*/
 if ((fptr=fopen("a4.data", "r"))==NULL) { /* Open file */
 printf("\nCan not open a4.data for read");
 exit(-1);
 }
 while (fread(&std, sizeof(Score), 1, fptr)) /* Read data */
 printf("\nID [%s], score : %.2f", std.id, std.score); /* Display data */
 fclose(fptr); /* Close file */
 return(0);
}
```

---

---

**แบบฝึกหัดบทที่ 8**

1. เขียนโปรแกรมเพื่อทำการรับค่าสตริงจากแป้นพิมพ์ เพื่อทำการบันทึกข้อมูลลงแฟ้มข้อมูลชื่อ Exam10.dat โดยกำหนดให้สตริงที่รับมีความยาวไม่เกิน 100 ตัวอักษร โดยใช้ฟังก์ชัน fputc()
2. เขียนโปรแกรมเพื่อทำการคัดลอกแฟ้มข้อมูล Exam10.dat ในข้อ 1. ไปจัดเก็บในแฟ้มข้อมูลใหม่ชื่อว่า Test1.dat
3. เขียนโปรแกรมเพื่อทำการคัดลอกข้อมูลจากแฟ้มข้อมูล Test1.dat และ Exam10.dat ไปเก็บไว้ในแฟ้มข้อมูลใหม่ชื่อว่า Test3.dat โดยมีเงื่อนไขในการคัดลอกแฟ้มข้อมูลดังนี้ ทำการอ่านข้อมูลที่ละ 1 อักขระจากแฟ้มข้อมูล Test1.dat จากนั้นบันทึกลงในแฟ้มข้อมูล Test3.dat จากนั้นทำการอ่านข้อมูลอีก 1 อักขระจากแฟ้มข้อมูล Exam10.dat ทำการบันทึกลงในแฟ้มข้อมูล Test3.dat ทำการอ่านข้อมูลสลับไปมาระหว่างแฟ้มข้อมูล Test1.dat และ Exam10.dat เพื่อบันทึกลงในแฟ้มข้อมูล Test3.dat หากข้อมูลที่อ่านในแฟ้มข้อมูลใดแฟ้มข้อมูลหนึ่งหมดก่อน ให้ทำการอ่านข้อมูลในแฟ้มข้อมูลที่เหลือเพื่อทำการบันทึกข้อมูลต่อไปจนหมดแฟ้มข้อมูล
4. ให้สร้างเท็กซ์ไฟล์ เก็บข้อมูลของเมตริกซ์ 2 เมตริกซ์ ซึ่งมีขนาด 3x3 อ่านข้อมูลจากแฟ้มดังกล่าวนำมาเก็บในอาเรย์ 2 มิติ จำนวน 2 อาเรย์แล้วหาผลบวกของอาเรย์ทั้งสอง ให้แสดงผลและบันทึกผลลัพธ์ที่ได้ในลักษณะเท็กซ์ไฟล์
5. เขียนโปรแกรมเพื่อรับข้อมูลสังกัดและคะแนนสอบของนักศึกษาจำนวน N คน และบันทึกข้อมูลไว้ในแฟ้ม student1.dat ให้ใช้คำสั่ง fprintf() ในการบันทึกข้อมูล โดยกำหนดให้ข้อมูลสังกัดใช้เลขรหัส 1 แทนนักศึกษาคณะวิทยาศาสตร์ และเลขรหัสอื่น ๆ แทนนักศึกษาคณะอื่น และคะแนนสอบของนักศึกษามีค่าอยู่ในช่วง 0 ถึง 100 คะแนน
6. เขียนโปรแกรมเพื่อคำนวณเฉลี่ยของคะแนนสอบของนักศึกษา แยกตามประเภทของนักศึกษา คือ นักศึกษาสังกัดคณะวิทยาศาสตร์ และนักศึกษาอื่น ๆ โดยอ่านข้อมูลจากแฟ้ม student2.dat ในข้อ 5 ให้อ่านข้อมูลสังกัดของนักศึกษาและคะแนนสอบไว้ในอาเรย์ของโครงสร้างให้เรียบร้อยก่อน แล้วจึงหาคะแนนเฉลี่ยของนักศึกษาสังกัดคณะวิทยาศาสตร์ และนักศึกษาสังกัดคณะอื่น
7. เขียนโปรแกรมเพื่อรับข้อมูลนักศึกษาจำนวน N คน โดยเก็บรายละเอียดข้อมูล รหัสนักศึกษา คณะที่สังกัด และชั้นปี โดยบันทึกข้อมูลในแฟ้ม student2.dat โดยใช้คำสั่ง fwrite()
8. เขียนโปรแกรมเพื่อสร้างตารางแจกแจงความถี่ของจำนวนนักศึกษาโดยแยกตามคณะและชั้นปีที่นักศึกษาสังกัด โดยอ่านข้อมูลจากแฟ้ม student2.dat ในข้อ 7 โปรแกรมจะสร้างตารางแจกแจงความถี่ดังตัวอย่าง

|           | First Year | Second Year | Third Year | Fourth Year | >4  | Total |
|-----------|------------|-------------|------------|-------------|-----|-------|
| Faculty 1 | 1500       | 1200        | 1150       | 1100        | 20  | 4970  |
| Faculty 2 | 3500       | 3300        | 3200       | 3100        | 100 | 12200 |
| ...       |            |             |            |             |     |       |
| Total     | ...        | ...         | ...        | ...         | ... | ...   |

ให้แสดงตารางแจกแจงความถี่บนจอภาพและเก็บลงแฟ้มข้อมูลใหม่ในลักษณะของเท็กซ์ไฟล์โดยใช้คำสั่ง fprintf()

9. เขียนโปรแกรมเพื่ออ่านข้อมูลจากแฟ้ม student2.dat ในข้อ 7 ให้อ่านข้อมูลเฉพาะนักศึกษาในลำดับที่ 5, 10, 15, ... ขึ้นมาแสดงบนจอภาพทั้งหมดจนกว่าจะหมดแฟ้มข้อมูล โดยใช้คำสั่ง fseek ช่วยในการระบุตำแหน่งของข้อมูลที่ต้องการ

## บรรณานุกรม

1. สมจิตต์ ลิขิตถาวร. *การเขียนโปรแกรมภาษาปาสคาล* : แผนกตำราและคำสอน มหาวิทยาลัยกรุงเทพ, 2543.
2. Brian W.Kernighan and Dennis M. Ritchie. *The C Programming Language* : Prentice Hall International Ltd., 1989.
3. C. Thomas Wu. *An Introduction to Object-Oriented Programming with Java* : McGraw-hill International Editions.,1999.
4. H.M. Deitel and P.J. Deitel. *Java How to Program, Third Edition* : Prentice Hall International Ltd., 1999.
5. Kenneth A.Barclay. *ANSI C Problem Solving and Programming* : Prentice Hall International Ltd., 1990.
6. M.Tim Grady. *Turbo C Programming Principles and Practices* : McGraw-hill International Editions.,1989.

